Volume II

Ada Implementation Guide

Software Engineering With Ada

April 1994

DEPARTMENT OF THE NAVY

Naval Information Systems Management Center

Contents

VOLUME II

St #A, Auth: AJPO (Mr. Currie Colket - 602-3968) Telecon, 7 Jul 94
CB

List of Figures and Tables

Figures

Tables

Appendix A
HELPFUL SOURCES

This appendix provides sources to help the Department of the Navy (DON) Program Manager become knowledgeable about Ada-related issues. Information is provided on several Government sources, including the Ada Information Clearinghouse (AdaIC), which is sponsored by the Ada Joint Program Office (AJPO) and other sources. The sources listed are not exhaustive, and the information regarding these sources may have changed since the publication of this document. DON does not endorse these sources, and Department of Defense's (DOD's) use of Ada does not imply in any manner that the DOD endorses or favors any commercial Ada product. These products are listed to inform Program Managers of what is available. Program Managers must use their own judgments about the value of the services. Additional sources can be added to this list for future editions of this guide by contacting the DON Ada Representative.

A.1 GOVERNMENT SOURCES

Government sources are organized into seven categories: organizations, training, publications, bulletin boards, repositories, conferences and special interest groups, and operational development support tools. The type of information contained in each of the categories is as follows:

- **Organizations**—DOD, DON, and Marine Corps organizations that focus on Ada policy, technical guidance, and programs with DON-wide applicability
- **Training**—sources of training and information about training for various types of personnel
- **Publications**—sources of newsletters and other publications
- **Bulletin Boards**—sources that maintain a public bulletin board directed at the Ada community
- **Repositories**—sources of reusable components and libraries
- **Conferences and Special Interest Groups**—information on regularly scheduled expositions, workshops, and symposia as well as conferences
- **Operational Development Support Tools**—information on environments and tools currently used by the DOD community

The amount of Ada-related information available from these sources is too vast to reproduce in this appendix. However, an address, telephone number, electronic mail address (if available), and short description are provided for each source. Often, the easiest way to obtain information from these sources is by electronic mail, but contact by mail, telephone, or facsimile is also possible.

A.1.1 Organizations

Ada Joint Program Office
1211 South Fern Street
Room C107
Arlington, VA 22202
(703) 614-0209
DSN: 224-0208

The Ada Joint Program Office (AJPO), which consists of a deputy director from each Service and a chairperson, is responsible for managing the effort to implement, introduce, and provide life-cycle support for the Ada programming language. The AJPO sponsors AdaIC, a primary source of Ada information.

Ada 9X Project
Project Manager
Phillips Laboratory/VTES
3550 Aberdeen Avenue, S.E.
Kirtland AFB, NM 87117-5776
(505) 846-0461
Internet: anderson@plk.af.mil

This project is responsible for revisions to the American National Standards Institute/Military Standard (ANSI/MIL-STD)-1815A to reflect current essential requirements with minimum negative impact and maximum positive impact on the Ada community.

Ada Board
Ada Joint Program Office
1211 South Fern Street
Room C107
Arlington, VA 22202
(703) 614-0209

The Ada Board provides AJPO with balanced advice and information on the technical aspects related to official interpretations of the Ada language standard and on issues associated with Ada validation and software environment activities.

Ada Information Clearinghouse
AdaIC
P.O. Box 46593
Washington, D.C. 20050-6593
(703) 685-1477
1-800-232-4211 (1-800-AdaIC-11)
Internet: adainfo@ajpo.sei.cmu.edu
FAX: (703) 685-7019
Compuserve: 70312,3303

Ada Joint Program Office
OUSD(A)/DDRE/AJPO
Room 3E118, The Pentagon
Washington, D.C. 20301-3081
(703) 614-0215
FAX: (703) 685-7019

The latest information about Ada is available to you free of charge from AJPO's Ada Information Clearinghouse (AdaIC). The AdaIC makes available information on a variety of topics ranging from the use of Ada within DOD and industry to tools and compilers for Ada developers, and from DOD policies regarding Ada to reusable Ada software.

The AJPO sponsors the AdaIC. The AJPO is responsible for informing the community about Ada, facilitating the language's implementation in the services, and maintaining the integrity of the language.

The telephone hotline numbers are 1-800-232-4211 outside the Washington, D.C. area, and (703) 685-1477 in the Washington, D.C. area. For answers to your Ada questions, call the AdaIC, Monday through Friday, from 8:00 a.m. to 5:00 p.m., Eastern Time.

Ada Validation Office
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria, VA 22311
(703) 845-6639

This office implements compiler validation policy and oversees development of the Ada Compiler Validation Capability (ACVC).

National Institute of Standards and Technology
Software Standards Validation Group
Building 225, Room A-266
Gaithersburg, MD 20899
(301) 975-3274
Attn.: Arnold Johnson

The National Institute of Standards and Technology (NIST) provides Federal Information Processing Standards (FIPS) for the Ada language. NIST also is an Ada validation facility and coordinates with AJPO for conformance testing, policies, and procedures.

DON Software Executive Official
Commander, Naval Information System Management Center
Crystal Plaza 5, Room 334
2211 Jefferson Davis Highway
Arlington, VA 22202
(703) 602-2103

The DON Software Executive Official (SEO) is the point of contact for all DON software and software-related issues.

DON Ada Representative
AST Software and Systems
Naval Information System Management Center (NISMC)
Building 166, Washington Navy Yard
Washington, D.C. 20374
(202) 433-4903/3499

This office is the point of contact for all Ada and Ada-related issues.

Space and Naval Warfare Systems Command
Code 224-1
5 Crystal Park
Suite 700
Washington, D.C. 20363-5100
(703) 602-9188

The Space and Naval Warfare Systems Command (SPAWAR) is the point of contact for DOD-STD-2176A, Defense System Software Development; computer resources management and interface standards for weapon systems applications (Secretary of the Navy Instruction [SECNAVINST] 5200.32A and Secretary of the Navy Note [SECNAVNOTE] 5200); and Navy representation on the Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resources Management (JLC-JPCG-CRM).

Next Generation Computer Resources
Space and Naval Warfare Systems Command
Code 311-2
5 Crystal Park
Suite 700
Washington, D.C. 20363-5100
(703) 609-9096

This office is the point of contact for all next generation computer issues.

Commander, Naval Computer and Telecommunications Command (COMNAVCOMTELCOM)
Ada Program Manager
4401 Massachusetts Avenue, N.W.
Washington, D.C. 22036-5460
(202) 282-0221
DSN: 292-0221
FAX: (202) 282-2684

This command is the headquarters for the Naval Computer and Telecommunications Stations (NCTSs). Through the Ada Technical Support Bulletin Board, the Naval Computer and Telecommunications Command (NAVCOMTELCOM) provides support for Ada projects and technical information to the Ada community at large. NCTC chairs the AdaSAGE Configuration Management Board, manages the Navy-wide Reuse Center, and publishes CHIPS.

Commandant of the Marine Corps
Director (CTAE-13)
MARCOR COMTELACT
3255 Meyers Avenue
Quantico, VA 22134-5048
(703) 640-4897
Internet: depasquale@mqgl.usmc.mil

This source is the primary point of contact for Marine Corps Ada program development.

Naval Center for Cost Analysis
Head, Automated Information Systems Division
Pentagon
Room 4A538
Washington, D.C. 20350-1100
Attn.: Stephen Gross
(703) 746-2342
DSN: 286-2342
FAX: (703) 746-2390

The Naval Center for Cost Analysis (NCA) was established 6 August 1985 by decision of the Secretary of the Navy with the following mission: "To provide independent cost and financial analyses to support the Secretary of the Navy . . . [and to] Ensure credible cost estimates of the resources required to develop, procure, and operate military systems and forces in support of planning programming, budgeting and acquisition management."

NCA is a field office of the Assistant Secretary of the Navy (Financial Management). It is located in the Crystal City area of Arlington, Virginia, near the Pentagon. NCA supports the Office of the Secretary of Defense (OSD) in satisfying Title 10 U.S. Code §2434, which requires independent life-cycle cost estimates, including the cost of research and development, procurement, and operations and support of major weapons systems such as ships, aircraft, missiles, and electronic systems. NCA also conducts financial analyses of defense contractors and economic analyses of acquisition issues.

Software Technology Support Center
Ogden Air Logistics Center
TISAC
Hill AFB, UT 84056

The Software Technology Support Center (STSC) acts as a focal point for the U.S. Air Force on software tools, methods, and environments. Its activities include a bulletin board, an annual software conference, a monthly newsletter, consulting services, reports on various software topics, information on software repositories, and other software support services. The STSC provides quantitative evaluations of technology, tailored to specific customer requirements, on a fee-for-service basis. The STSC also has several technology insertion projects and is working directly with specific customers in selecting new technology and inserting the technology. Among the reports on software topics are a software manager's guide, a project management technology report, a reengineering technology report, and tool reports on various domains from Computer-Aided Software

Engineering (CASE) tools to documentation tools. Information on software repositories is a special examination being conducted by the STSC. It will result in a report and a number of activities with various customers.

Software Engineering Institute
Customer Relations
Carnegie-Mellon University
Pittsburgh, PA 15213-3890
(412) 268-5800
FAX: (412) 268-5758

The Software Engineering Institute (SEI) is a Federally Funded R[illegible]rch and Development Center (FFRDC) sponsored by DOD through the Advanc[illegible]esearch Projects Agency (ARPA). The SEI provides leadership in advancing the [illegible]te of the practice of software engineering to improve the quality of systems that depend on software. The SEI's four areas of focus are software process, software risk management, real-time distributed systems, and software engineering techniques. To increase the number of highly qualified software engineers, the SEI also seeks to improve software engineering education within academia, Government, and industry.

In response to computer security threats, ARPA established the Computer Emergency Response Team (CERT) Coordination Center at the SEI to support Internet users. The members of the CERT Coordination Center work with the Internet user community and technology producers to address and prevent computer emergencies.

To accelerate the dissemination of new technologies and methods, the SEI offers U.S. organizations from academia, Government, and industry several methods of interacting with the institute. Information on the subscriber program, technical reports, continuing education courses, and symposia may be obtained by calling or writing to the Customer Relations Office.

Software Technology for Adaptable, Reliable Systems (STARS)
801 North Randolph Street, Suite 400
Arlington, VA 22203
(703) 351-5300
Internet: (for newsletter) newsletter@stars.ballston.paramax.com
(for ASSET) STARSBBS@source.asset.com
INFO@source.asset.com
For more information:
Joel Trimble—E-mail trimble@stars.ballston. paramax.com or above address

The STARS Program is a major ARPA/Software and Intelligent Systems Technology Office (SISTO) effort to provide more capable, efficient and productive methods of developing software for DOD. STARS' goals are to (1) improve productivity; (2) improve quality and reliability; (3) promote development and application of reusable software; and (4) promote adaptability, evolvability, and interoperability through the use of standard interfaces and open architectures, both of the application software and of the Software Engineering Environments (SEEs), which support that application software.

The technology areas STARS supports include SEE frameworks, software reuse mechanisms, and tailorable software process models. STARS includes the national Asset Source for Software Engineering Technology (ASSET) software reuse library effort.

STARS maintains an affiliates program to provide an opportunity for the DOD software community to participate in STARS technical activities. The three levels of affiliates (e.g., individual representatives of Government agencies, universities, vendors) are as follows:

- Information affiliates who receive the STARS newsletter, attend STARS conferences, (STARS 9X), have access to the STARS bulletin board, and attend STARS technology demonstrations at the STARS Technology Center

- Technology transition affiliates who attend technical exchange working group meetings; voluntarily participate in selected receptor organizations; and are involved in alpha/beta testing, feedback, lessons learned, and product evolution

- Prime affiliates who work directly with STARS prime contractors in relevant technical activities such as technology transition, production evaluation, and tool development.

The parent sponsoring organizations are responsible for labor, travel, and other expenses associated with participating in the affiliates program.

A.1.2 Training

Ada Language System/Navy
ALS/N Training
NCCOSC RDTE DIV 924
53560 Hull Street
San Diego, CA 92152-5800
(619) 553-0949

Training provided at this source includes Ada Language System/Navy (ALS/N) courses pertaining to Ada/M (UYK-44(V) target), Ada L (AN/UYK-43(V)), PPI (AN/AYK-14 (V) target), and Common Ada Baseline/Project Support Environment (CAB/PSE) tools (a VAX/VMS host with associated PSE tools). Training is available by request in conjunction with ALS/N quarterly meetings, and on-site training is available.

Ada Software Engineering Education and Training Team
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria, VA 22311
Attn.: Cathryn McDonald
(703) 845-6626
(719) 472-3131
(202) 282-0833

The Ada Software Engineering Education and Training (ASEET) team is composed of representatives from the Army, Air Force, Navy, Marine Corps, other DOD agencies, and academia. The team conducts workshops and symposia for Ada educators within DOD and academia and coordinates the activities of DOD organizations engaged in meeting the Ada education and training needs. The team is also available, as funding permits, to present 1- day or half-day introductory and advanced tutorials on Ada and software engineering. It also conducts an annual Ada symposium that focuses on education and training issues. An ASEET resource library of educational materials is located at AJPO. ASEET also has established an Ada Materials Library that contains copies of all team-developed tutorials and many other Ada documents and textbooks.

Team members are located all across the continental United States and could serve as local resources to answer questions on Ada or to direct inquiries to non-local sources.

The team has tri-Service representation on four working groups that address education and training on requirements, courseware, professional development, and coordination. Major tasks include the following:

- Identifying education and training requirements within DOD
- Conducting Ada research projects
- Managing Ada course materials
- Performing Ada certification study
- Providing a database of ASEET research data and final reports
- Providing a DOD focal point for Ada software engineering education and training.

AdaSAGE
Department of Energy
Idaho National Engineering Laboratory
Idaho Falls, Idaho
(208) 526-0656

The Idaho National Engineering Laboratory (INEL) offers training and support through a hot line subscription service.

Air Force Institute of Technology
Wright Patterson AFB
Ohio 45433
(513) 255-6027

The Air Force Institute of Technology currently teaches the following courses that use Ada as an implementation language:

- **Introduction to Data Structures and Program Design**–Principles and methodologies used to design and implement small programs.
- **Advanced Information Structures**–Structure of data and the efficient and effective manipulation (algorithms) of such structures.
- **Operating Systems**–Concepts and principles of computer operating systems. The objective is to give the student an understanding of operating systems and the necessary skills to evaluate and trade-off desirable features of operating systems given specific user and resource requirements.
- **Software Analysis and Design**–Examination of the object-oriented paradigm and the formal specification of software. Object-oriented design and formal specification.
- **Software Systems Engineering**–Basic principles and techniques underlying object-oriented and object-based generation of software.
- **Analysis and Maintenance of Software Systems**–Basic principles and techniques underlying the measurement and analysis of software systems as well as the principles and techniques underlying the maintenance of existing software.
- **Algorithms for Parallel Processing**–Understanding of classical results for parallel design and analysis and provides practical insights into efficient and effective implementation on contemporary parallel computational machines.

- **Compiler Theory and Implementation**–Theoretical foundation of formal languages and compiler theory.

- **Principles of Embedded Software**–Mathematical and computer science principles for the specification, design, implementation, and analysis of embedded software systems.

- **Design of Fault Tolerant Software**–Basic mathematical principles, data structures, and algorithms associated with the design of fault-tolerant software systems.

- **Formal-Based Methods in Software Engineering**–The mathematical and computer science theory used as the basis for developing formal-based methods for specifying, generating, and validating or verifying software.

Common Ada PSE Interface Set (CAIS)

CAIS-A
Commander
Naval Ocean Systems Center
271 Catalina Boulevard
San Diego, CA 92152-5000
Attn.: CAIS-A Training Coordinator
(619) 553-6858

A 5-day training class is available that provides hands-on experience for Ada tool designers. (Knowledge of the Ada programming language is a prerequisite.) Training will be available on a VMS system and on a Sun 3 running under UNIX. Also available are the following:

- CAIS-A Self-Study Guide
- CAIS-A Tool Writers Guide.

Computer Sciences School

Head, Application Programming Instructional Department (APID)
Marine Corps Combat Development Command (MCCDC)
3255 Meyers Avenue
Quantico, VA 22134-5051
(703) 640-2962
DSN: 278-2962
(703) 640-3759

The Computer Sciences School (CSS) currently offers the following Ada-related training courses:

- Entry-level Ada Programming Course. This course is designed for the student who has no previous programming experience or training. Programming concepts and principles are taught along with the basics of Ada programming. Additionally, subjects, such as TSO, JCL, and IBM utilities, are taught to expose students to tools they may need as programmers. This course is 8 weeks long (41 training days) and is offered four or five times a year.

- Ada Programming Course. This course is designed to teach the basics of the Ada programming language to programmers who are currently working in a language other than Ada. However, programming experience is not a requirement to attend the course. The course is 4 weeks long (20 training days) and is offered three times a year.

- Advanced Programming Techniques (APT) Course. This course is not a typical programming course. No syntax is taught. Instead, the course teaches software project management principles in conjunction with software analysis and Object-Oriented Design (OOD) techniques. This course is 3 weeks long (15 programming days) and is offered three times a year.

Personnel interested in attending a course taught at CSS should contact the Academics Officer, Training and Operations Section (TOPS), DSN 278-2891 or COMM (703) 640-2891, *for specific information on registering for a course.*

Other Points of Contact (POCs) are:

- Marine Corps—Steve Bruzek, 4000 MOS Sponsor at DSN 241-3593 or COMM (703) 614-3593

- Navy—Navy DP Detailer at DSN 223-3537 or COMM (703) 693-3537

- All civilians and other service personnel—SSgt Riegal, Quota Control Manager, Training and Education Division at DSN 278-3071 or COMM (703) 640-3071.

Computer Science School
Chief of Operations
Army Computer Science School
U.S. Army Signal Center & Fort Gordon
Fort Gordon, GA 30905
(706) 791-2586

The Army Computer Science School currently offers a 2-week (10-training-day) course in Structured Programming in Ada for Active and Reserve Component commissioned and

warrant officers and non-commissioned officers in grades E-6 and above and DOD civilians in grades GS-07 and above, Active-duty or Reserve Data Processors should contact the Navy DP Detailer, NMPC-406, DSN 223-3537 or COMM (703) 693-3537. Other Navy personnel and civilians interested in enrolling should contact Navy's ATTRS POC, Ms. Holder, OPNAV-112G1, DSN 225-8665 or COMM (703) 695-8665 to enroll through ATTRS, or contact Chief of Operations, Army Computer Science School, DSN 780-2326 or COMM (706) 791-2326. The Army Computer School also teaches Ada as part of the Systems Automation Officer course, and students use Ada in their software engineering project.

National Audiovisual Center
8700 Edgeworth Drive
Capitol Heights, MD 20743-3701
Attn.: Customer Service Department
(301) 763-1891
FAX: (301) 763-6025

A series of Ada training tapes sponsored by the AJPO is available for purchase through the National Audiovisual Center of the Department of Commerce. The tapes include the following:

- Introduction to Ada (3 tapes; about 3 hours; order no. A18336; $150)
- Ada from a Management Perspective (2 tapes; about 80 minutes; order no. A18337; $100)
- Software Engineering in Ada (19 tapes; about 8 hours, 20 minutes; order no. A18338; $500).

Additional information on these tapes is available from the AdaIC.

National Defense University
Information Resources Management Curriculum
Fort McNair, Washington, D.C.
(202) 287-9340

Ada is examined in the Programming Languages course in the Advanced Management Program.

Naval Postgraduate School
Monterey, CA 93943-4444
(408) 656-2591

The Naval Postgraduate School teaches Ada in the following courses:

- Structured Programming with Ada
- Data Structures
- Software Methodology (the process of software development)
- Software Engineering (formal methods)
- Software Engineering with Ada (task, real-time issues)
- Computers in Combat Systems
- Software Tools and Environments.

Software Engineering Institute
Carnegie-Mellon University
Pittsburgh, PA 15213-3890
(412) 268-5800
FAX: (412) 268-5758
Internet: education@sci.cmu.edu

The SEI has collected six software "artifacts," called EM1-EM6, targeted at teaching software engineering. Artifact EM1, for example, is a 10,000-line Ada style checker packaged with exercises to teach software maintenance. SEI also produces many technical reports, including the following, which are highly recommended:

- *Ada Adoption Handbook: A Program Manager's Guide*
- *Ada Adoption Handbook: Compiler Evaluation and Selection.*

United States Air Force Academy
Headquarters, USAFA/DFCS
2354 Fairchild Drive, Suite 6K41
USAFA, CO 80840
(719) 472-3131
FAX: (719) 472-3338
Internet: dcook@kirk.usafa.af.mil
POC: CAPT David Cook

The U.S. Air Force Academy teaches Ada to computer science majors in the Foundations of Computer Science course. Majors also use Ada in the Programming Languages course and the Algorithms and Data Structures course. Additionally, a 2-week course open to anyone is taught during the summer (June/July). Space is limited; therefore, early registration is advised. There is no charge for the course, but all students must pay their own travel and per-diem costs.

United States Air Force Technical Training School
Keesler Air Force Base
Biloxi, MS
(601) 377-5379
DSN: 597-5379

The U.S. Air Force Technical Training School teaches Ada in its Fundamentals of Ada Programming/Software Engineering course and its Ada Applications Programmer course.

United States Army Engineering College
Rock Island Arsenal
Rock Island, IL 61299-7040
Attn.: AMXOM-RS
(309) 782-0488/0489/0487

The U.S. Army Engineering College provides a 2-week Ada overview free to Government employees (Course No. AMEC-140). Additionally, four to five courses that run 41 training days per year include entry-level Ada programming. They are currently developing software project management geared toward OOD in FY94.

United States Military Academy
Westpoint, NY 10996
(914) 938-5607
FAX: (914) 938-5438
Internet: DT2283@eecs1.eecs.usma.edu
POC: CAPT Crabtree

The United States Military Academy first introduces Ada to computer science majors in their second year in the course Analysis of Programming Languages. The following year, they take Software Engineering with Ada for a full semester. This course introduces the students to software engineering and focuses on how Ada supports the principles and goals of software engineering. The course treats software engineering concepts in detail. The OOD paradigm is introduced and practiced in programming assignments.

United States Naval Academy
Computer Science Department
572 Holloway Road
Annapolis, MD 21402
(410) 267-2797/8
FAX: (410) 267-2686
Internet: eun@csserrera.scs.usna.navy.mil

DSN: 281-3007
POC: Dr. E.K. Park

The U.S. Naval Academy teaches Ada to computer science majors in their senior year in the Software Engineering and Advanced Software Engineering courses.

* * * * *

All of the major Ada compiler vendors have training available directly through their offices.

A.1.3 Publications

Defense Technical Information Center
Cameron Station
Alexandria, VA 22304-6145
Attn.: FDRA
(703) 274-7633

The Defense Technical Information Center (DTIC) distributes documents only to military, Government, or defense contractors who are registered users of DTIC. Most unclassified documents that are submitted to DTIC are also forwarded to the National Technical Information Service (NTIS) and are available to the public.

National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161
(703) 487-4650

NTIS, a self-supporting agency of the U.S. Department of Commerce, provides free publications and directories of Government databases and software components. The Application Portability Profile (APP) and FIPS are available from NTIS.

Standardization Documents Order Desk
Building 4, Section D
700 Robbins Avenue
Philadelphia, PA 19111-5094
Special Assistance Desk: (215) 697-2179
DSN: 442-2179
Customer Service: (215) 697-2667

This desk is the central distributor of all military standard documents, including the standard for the Ada language reference manual (ANSI/MIL-STD-1815A-1983). DOD standards, specifications, handbooks, and data items can be ordered by using the Telephone Order-Entry System (TOES). Access TOES by calling (215) 697-1187 (DSN 442-1187), Monday through Friday, 7:00 a.m. to 4:30 p.m. Eastern Time.

U.S. Government Printing Office
Superintendent of Documents
Washington, D.C. 20402-9371
(202) 783-3238

The Superintendent of Documents can provide the APP.

Ada 9X Publications
Phillips Laboratory/VTES
3550 Aberdeen Avenue, S.E.
Kirtland AFB, NM 87117-5776
(505) 846-0461

The Ada 9X Project Office maintains a mailing list for Ada 9X documents. To be placed on the mailing list or to receive hard copies of Ada 9X documents, send an E-mail message to the following address: keeneyr@plk.af.mil. For access to electronic versions of Ada 9X documents, leave a message at action@ajpo.sei.cmu.edu

Ada and C++
Software Technology Support Center
Ogden Air Logistics Center
TISAC
Hill AFB, UT 84056
(801) 777-7703

This report describes studies that compared Ada to C++. An electronic summary of this report is available on the AdaIC Bulletin Board. The report is also available through DTIC and NTIS.

Ada Information Clearinghouse Newsletter
AdaIC
P.O. Box 46593
Washington, D.C. 20050-6593
1-800-232-4211

The AdaIC quarterly newsletter contains current news from the AJPO about the Ada program, Ada conference reports, and articles on projects using Ada. If you would like to receive the newsletter, call the AdaIC and request a free subscription.

Ada Slices
MITRE Corporation
1120 NASA Road 1
Houston, TX 77057
(713) 335-8541

This newsletter is published by MITRE, an FFRDC. It is a product of the Association for Computing Machinery (ACM) Special Interest Group on Ada's (SIGAda's) Performance Issues Working Group (PIWG) and is available free of charge.

Ada Software Engineering Education and Training Public Report
Ada Software Engineering Education and Training Team
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria, VA 22311
Attn.: Resource Staff Member
(703) 845-6626

ASEET publishes a DOD ASEET Public Report annually. The report contains an update and description of the latest efforts of the ASEET Team in identifying training and education requirements within DOD and the methodology and materials needed to fulfill those requirements. Copies of the report may be obtained from the AdaIC.

Bridge
Eileen Forrester
Managing Editor, Bridge
Software Engineering Institute
Carnegie-Mellon University
Pittsburgh, PA 15213-3890
(412) 268-6377
Internet: bridge-editor@sei.cmu.edu

This magazine reports on SEI projects and activities. To obtain a subscription, send a written request to the editor.

CHIPS
9456 Fourth Avenue, Suite 200
Naval Computer and Telecommunications Area Master Station, Atlantic (NCTAMS LANT)
Norfolk, VA 23511-2199
(804) 444-8704
DSN 564-8704

CHIPS is a microcomputer magazine for mid-level users. It contains primarily product reviews, microcomputer contract information, and articles of general interest to the microcomputer community.

Crosstalk, The Journal of Defense Software Engineering
Software Technology Support Center
Ogden ALC/TISE
Hill AFB, UT 84056
Attn.: Customer Service
(801) 777-2237
DSN: 458-2237
FAX: (801) 777-8069
Internet: bbliss@oodis01.hill.af.mil

Crosstalk, The Journal of Defense Software Engineering, is published to help improve the effectiveness of software used by DOD. The journal provides information about software tools, methods, and environments for DOD software development and support activities, contractors who develop software for the military, and vendors who produce CASE tools for the defense market. *Crosstalk* frequently features articles on various aspects of Ada, from work on Ada 9X to techniques for converting from COBOL to Ada when reengineering management information systems. STSC distributes *Crosstalk* without charge to individuals actively involved in the defense software development process. To request to be added to the mailing list, write to the above address. To request other reports on software development tools and other topics, call (801) 777-7703 or DSN 458-7703.

DACS Newsletter
Barbara Radzisz
Editor, DACS Newsletter
Data & Analysis Center for Software
P.O. Box 120
Utica, NY 13503
(315) 734-3696

DACS Newsletter is the current awareness publication of the Data and Analysis Center for Software (DACS). It serves as a centralized source for current, readily available data and information on software engineering and software technology.

Institute for Defense Analyses
Computer & Software Engineering Division
1801 North Beauregard Street
Alexandria, VA 22311
(703) 845-2000 (General)
(703) 845-2059 (References)

The institute is an FFRDC the primary sponsor of which is the Office of the Secretary of Defense (OSD). All publications prepared by the institute are available through DTIC or NTIS.

High Order Language Control Facility Ada-JOVIAL Newsletter
645 C-CSG/SCSL
Wright-Patterson AFB, OH 45433-5707
(513) 255-4472
DSN 785-4472
langed@ss1.sews.wpafb.af.mil

To support the DOD and Air Force standardization efforts, information is disseminated about Ada and JOVIAL (J73), standardization and language control activities, training, compilers, compilers and tools, development efforts, applications, and user services.

NISMC Newsletter
Ms. Alcinda Wenberg
NISMC
Building CP5
Jefferson Davis Highway
Arlington, VA 22203
(703) 602-2542

This monthly newsletter provides information on Naval Information System Management Center (NISMC) initiatives, status of DON policy, and upcoming DON activities.

STARS Newsletter
801 North Randolph Street
Suite 400
Arlington, VA 22203
(703) 351-5300
newsletter@stars.ballston.paramax.com

The STARS newsletter contains articles covering software reuse technology, software process technology, and software engineering environment framework technology. It is published twice per year and is free of charge.

A.1.4 Bulletin Boards

Ada 9X Project
Project Manager
Phillips Laboratory/VTES
3550 Aberdeen Avenue, S.E.
Kirtland AFB, NM 87117-5776
(505) 846-0461
anderson@plk.af.mil

Information can be obtained from the ADA 9X Bulletin Board by calling 1-800-Ada-9X25 or (301) 459-8939 or by using the electronic address shown above. To access the bulletin board by modem, use the following settings:

- Baud rate = 300, 1200, or 2400
- Parity = none
- Data bits = 8
- Stop bits = 1

AJPO Host and Ada Information Clearinghouse Bulletin Board
1211 South Fern Street
Room C107
Arlington, VA 22202
(703) 614-0209

The AJPO sponsors two bulletin boards that serve as a primary source for Ada information. The *ajpo* host is accessible electronically on the Internet. The AdaIC Bulletin Board (AdaIC BB) is accessible by modem. Section A.2 provides details for accessing either bulletin board. The *ajpo* host and the AdaIC BB contain duplicate information.

Ada Technical Support Bulletin Board Service
Naval Computer and Telecommunications Area Master Station Atlantic
(NCTAMS LANT)
Norfolk, VA
(804) 444-7841
DSN 564-7841

To access by modem, use the following settings:

- Baud rate = 300, 1200, or 2400
- Parity = none
- Data bits = 8
- Stop bits = 1

NAVCOMTELCOM sponsors an Ada Technical Support Bulletin Board System (BBS) maintained by NCTAMS LANT.

The main purpose of the BBS is to offer microcomputer Ada programmers in the joint Services, Government contractors, and the academic community a means for obtaining answers to their questions about the Ada programming language. The BBS is targeted to programmers in the AIS domain and to software creation on these systems.

The BBS offers several services:

- *Ada Question and Answer Service.* BBS users can ask questions about the Ada language and extensions (e.g., pragmas) that might be included in a particular implementation. Additionally, user code can be uploaded for evaluation. Such evaluations can include checks for proper usage of Ada features, Ada style, and compilation errors that will not go away.

- *Compiler Vendor Comment Service.* BBS users can comment on DOS-based Ada implementations. Comments can report either problems with existing implementations or suggest enhancements that would benefit the DOS-Ada community. These comments will be provided to the appropriate compiler vendors. The goal is to use this service to improve DOS-based Ada compilers. A secondary benefit is to make potential users aware of possible problems with particular DOS-based Ada compiler implementations.

- *Ada Limited Debugging Assistance.* BBS users can upload small amounts of code to be debugged. The submitted code must be limited to a few program units.

- *AdaSAGE Question and Answer Service.* Many DOS-Ada application developers use AdaSAGE for Database Management System (DBMS) functions. This service is for AdaSAGE users. Users will be able to ask one another questions about AdaSAGE.

- *AdaSAGE Comment Service.* BBS users can comment on Ada application development using AdaSAGE. Comments can either report problems with AdaSAGE or suggest enhancements that would benefit future versions of AdaSAGE. These comments will be collected and presented periodically at AdaSAGE enhancement meetings. Users may also request AdaSAGE enhancements.

- *Ada Example Set.* This collection of code shows Ada features. BBS users can download the code, study it, and ask questions about it. Users also can upload code that shows Ada features.

- *News.* The BBS will list Ada news, events, and interesting Ada products and their points of contact.

The service is free and available to the public. However, the limited debugging service is available to bona fide Government employees and their contractors.

STSC Bulletin Board System
Ogden Air Logistics Center
TISE
Hill AFB, UT 84056
(801) 774-6509
DDN: Telnet 137.241.33.1 or stscbbs.oo.aflc.af.mil

The STSC sponsors the Electronic Customer Services (ECS), which is divided into the Bulletin Board System (BBS) and the Central Database (CDB). The purpose of ECS is to present the latest software information and knowledge to software practitioners in the DOD, industry, and academia.

To access the ECS by modem, use the following settings:

- Baud rate = 300, 1200, 2400, or 9600
- Parity = none
- Data bits = 8
- Stop bits = 1

The BBS offers information and news on a variety of software topics. The entries on the main menu are as follows:

- *Ada Information.* Presents the most recent Ada information, policy, news and trends.

- *USAF Software Policy and Regulations.* Abstracts applicable software policies and regulations including POC or author, office of responsibility, address, phone number, and latest date of publication.

- *Notes.* Allows users to leave a note, make a comment, or present their views to the STSC and other BBS customers on any software subject. The STSC will then put the notes into the appropriate BBS domain for subsequent viewing and comment.

- *Crosstalk, The Journal of Defense Software Engineering.* Lists every issue of the STSC's journal of software engineering. Hard copies are available on request.

- *DOD Corporate Information Management (CIM).* Presents most recent information from high-level DOD software management.

- *Information Technology Policy Board (ITPB).* Presents most recent information on and activities of this CIM-sponsored board.

- *Conferences, Meetings, Seminars.* Contains a comprehensive calendar of software activities sponsored by the Government, industry, academia, and international agencies.

- *Other Bulletin Board Systems.* Lists BBSs sponsored by the Government, industry, and academia.

- *Other Software Organizations.* Lists software organizations sponsored by the Government, industry, and academia.

- *Software Technical Domains.* Contains domains such as system simulation, requirements tracing, design, coding, testing and integration, documentation, project management, configuration management, quality, metrics, environments, and databases.

- *Software Engineering Topics.* Includes education, goals, and logistics (softlog), methods, metrics, processes, quality reengineering, and reuse.

- *Software News*. Gives access to an electronic newspaper featuring news and information from the software community.

- *Software Periodicals and Books*. Presents a list of the software periodicals, newspapers, journals, magazines, and newsletters published by the Government, industry, and academia.

- *Software Technology Conference*. Presents news about the annual Software Technology Conference (STC) held each April in Salt Lake City, Utah.

- *STSC Documents*. Lists all of the documents and reports generated or sponsored by the STSC. Hard copies will be sent on request.

- *White Papers*. Contains technical proposals from the software community at large.

- *Upload Files to the STSC*. Contains instructions on how to upload files to the STSC.

The CDB is a repository of software tool information. It gives descriptions of tools, addresses of vendors, and the ability to query for selected tool domains.

Cost Bulletin Board System
Air Force Cost Center
1111 Jefferson Davis Highway, Suite 403
Arlington, VA 22202
(703) 746-5840
DSN: 286-5840
Air Force Cost Bulletin Board POC: Ray Scheuring
(703) 746-5875 or 5876
1-800-344-3602

The Cost BBS provides an automated means of exchanging information, and uploading and downloading cost models and factors. If you have models or information you would like to have included, contact the system operator or leave a message on the bulletin board.

To access the Cost BBS, you must have an IBM-compatible microcomputer, communications software that allows XMODEM (checksum), XMODEM (CRC), ASCII, YMODEM or KERMIT file transfer protocols, and a Hayes-compatible modem. Communications settings are as follows:

- Baud rate = 1200 or 2400
- Parity = none
- Data bits = 8
- Stop bits = 1

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
(703) 321-8020

The NTIS Bulletin Board provides information on Computer-aided Acquisition and Logistics Support (CALS), CIM (e.g., Technical Reference Model), and more. Communications settings are as follows:

- Baud rate = 1200 or 2400
- Parity = none
- Data bits = 8
- Stop bits = 1

A.1.5 Repositories

Ada Software Repository
ada-sw-request@wsmr-simtel20.army.mil

The Ada Software Repository (ASR) contains Ada programs, software components, and educational material that has been established on the Defense Data Network (DDN). This repository has been accessible to any host computer on the DDN since 26 November 1984.

ASR is a free source of Ada programs and information. By employing the File Transfer Protocol (ftp) program, users of DDN hosts are able to scan the directories of the repository and transfer files to their hosts. If the files are Ada programs, they may then compile these programs and use them as they desire. Modifying these programs may be within their rights, and they may freely distribute these programs as they wish, subject to the restrictions specified in the prologue of each piece of software.

All members of the Ada community are encouraged to extract information and programs from the repository and to make contributions to it. The only restrictions that apply to access and use of this software are presented in the Distribution and Copyright section of the prologue associated with each piece of software.

ASR is one of several repositories located on the SIMTEL20 DDN host computer at White Sands Missile Range, New Mexico. SIMTEL20 is owned and operated by the Operations and Systems Integration Division of the Information Systems Command of the U.S. Army.

ASR maintains source code from approximately 10,000 Ada programs. These programs are maintained by the domains of Artificial Intelligence (AI), Benchmarks, Communications, Reusable Software Components, Documentation, Graphics, Project Management, Ada Software Development Tools, and many others. The ASR is available through ftp and on magnetic tape, floppy disk, and CD-ROM.

An introduction to the ASR can be obtained by using the following commands on a system that supports ftp on the DDN:

```
>ftp wsmr-simtel20.army.mil
when asked for login name, type in anonymous
when asked for password, type in your user-id
ftp>ls                        —provides listing of login directory
ftp>get SIMTEL20-ADA.INF      —copies file to your local directory
ftp>quit                      —returns control to UNIX
```

Tape copies are available from:

The DECUS Program Library
219 Boston Post Road BP02
Marlboro, MA 01752
(508) 480-3418

MS-DOS high-density diskette copies are available from:

Advanced Software Technology, Inc.
P.O. Box 937
Medford, NY 11763
(516) 289-6646

CD-ROM copies are available from:

ALDE Publishing
P.O. Box 35326
4830 West 77th Street
Minneapolis, MN 55435
(612) 835-5240
FAX: (612) 835-3401

An electronic mailing list exists on SIMTEL20 for those who are interested in accessing and contributing software to the ASR. To subscribe to this mailing list, send a request to the electronic mail address above.

Air Force Defense Software Repository System
SSC/SSB Building 856, Room 265
Maxwell AFB, Gunter Annex, AL 36114-5000
(205) 416-5857
DSN: 596-5857
FAX: (205) 416-5964

The Air Force Defense Software Repository System (AFDSRS) is a repository of Air Force and commercial reusable software assets, including functional requirements, design specifications, architectures, design diagrams, source code, documentation, and test suites. AFDSRS is accessible by modem or the DDN and is linked to the Defense Information Systems Agency (DISA) Center for Software Reuse Operations (CSRO) library, which offers access to all DOD components.

Associate Director, MCSD
AMSEL-RD-SE-BCS-MC (C2MUG)
Fort Leavenworth, KS 66027
AUTOVON: 552-7550
FTS: 753-7550
(913) 684-7550

The C2MUG Software Catalog for mathematics and various Ada functions is available to all echelons of the U.S. military and elements of the Federal Government. Software components are primarily for microcomputers.

Central Archive for Reusable Defense Software Program
CARDS
1401 Country Club Road, Suite 201
Fairmont, WV 26554
(304) 363-1731
CARDS Hotline: 1-800-828-8161 or (304) 367-0421
E-mail for Hotline: hotline@cards.com (Internet)
Cards Program Sponsor: ESC/AVS, Hanscom AFB, (617) 377-9369
DSN 478-9369

The Central Archive for Reusable Defense Software (CARDS) program is a concerted DOD effort to move advances in the techniques and technology of architecture-centered, domain-specific software reuse into mainstream DOD software procurements. CARDS

is applying the latest technology to provide an implementation framework for reuse libraries in domains of interest to the DOD. CARDS is currently applying the framework to a Command Center Domain Library. CARDS is working closely with the Portable Reusable Integrated Software Modules (PRISM) program, which is integrating Commercial-Off-The-Shelf (COTS) and Government-Off-The-Shelf (GOTS) products to perform 80% of the general functions of normal command center operations. The CARDS program is currently developing several reuse handbooks.

Command, Control, Communications, and Intelligence Reusable Software System

Mr. Ron Crepeau
NRaD
271 Catalina Boulevard
San Diego, CA 92152-5000
(619) 553-3990
crepeau@nosc.mil

The Command, Control, Communications, and Intelligence Reusable Software System (CRSS) is a repository of Navy Command and Control (C2) assets, including source and executable code, documentation, and graphical representations. The library has been developed under the Operations Support System (OSS) project to promote rapid prototyping of C2 systems. Replication of the CRSS is available on magnetic media or by modem.

Common Ada Missile Components Effort

Data and Analysis Center for Software
c/o Kaman Sciences Corporation
P.O. Box 120
Utica, NY 13503
(315) 336-0937

Common Ada Missile Packages (CAMP) are operational flight software parts written in Ada for tactical missiles. CAMP consist of 454 reusable Ada components. The software is distributed on ANSI standard labeled 9-track 1600-bits-per-inch tapes. Additionally, videotapes on Ada reuse are available, such as "Common Ada Missile Packages—Leading the Way in Software Reuse." This videotape provides an overview of Ada, software reuse, and the CAMP program.

Data and Analysis Center for Software
258 Genesee Street
Suite 101
Utica, NY 13502
(315) 734-3664

Although not an interactive repository, Data and Analysis Center for Software (DACS) provides several products and services. Of importance are the following: the Ada Compiler Evaluation System (ACES), (a set of Ada benchmarks), CAMP (a collection of reusable Ada packages), a set of benchmarks, and a cataloging facility in addition to various technical reports.

Defense Software Repository System
DISA/JIEO/CIM Software Reuse Program
500 North Washington Street, Second Floor
Falls Church, VA 22046
(703) 536-6900/7485

The DISA Joint Interoperability and Engineering Organization (JIEO) and CIM Software Reuse Program (SRP) is an element of the DOD Software Reuse Initiative under DISA/JIEO/CIM. The DISA/JIEO/CIM mission is to provide software reuse products and reusable software, training, and access to the Defense Software Repository System (DSRS). The SRP includes support of DOD Software Reuse Centers at the Service and agency levels throughout DOD to coordinate software reuse efforts and maximize cross-domain sharing.

National Aeronautics and Space Administration's AdaNet
AdaNet
c/o MountainNet
Eastgate Plaza, 2nd Floor
P.O. Box 370
Dellslow, WV 26531-0370
(304) 296-1458
1-800-444-1458

AdaNet's primary purpose is to increase U.S. productivity, economic growth, and competitiveness through development of a life-cycle repository for software engineering products, processes, interfaces, and related information services. AdaNet is sponsored by NASA, and there is no charge for an account.

AdaNet provides the following information and services:

- Access to Ada source code libraries
- Bibliographic references to Ada and software engineering publications
- Descriptions of public and commercial repositories of Ada software
- Directories of Ada and software engineering commercial products
- Electronic forums on topics such as software reuse and CALS
- Listings of international Ada professional organizations
- Monthly listings of relevant conferences and seminars
- References to public and private Ada information services.

Navy Wide Reuse Center

Project Manager, Navy Wide Reuse Center
Washington Navy Yard
Building 196
Code N53, Room 4508
Washington, D.C. 20374
POC: Angus Faust
(202) 433-0718
nhis.navy.mil

The Navy Wide Reuse Center (NWRC), which was dedicated on 16 March 1992, will provide a comprehensive reuse support environment for all Navy domains. The center will serve as a repository for all Navy reusable components and provide interfaces to other DOD and non-DOD repositories as well as information on commercially available reusable components. NWRC uses the DSRS hosted on a DEC MicroVAX computer. DSRS is accessible through DDN, modem dial-up, and selected Local Area Networks (LANs). An account on the system is required and should be requested through the above address.

Reusable Ada Products for Information Systems Development

U.S. Army
Army Reuse Center
Fort Belvoir, VA 22060-5456
Attn.: USAISEC Stop-H10
(703) 285-9007
DSN: 356-9007

The Reusable Ada Products for Information Systems Development (RAPID) Program is a total Ada software reuse program established at the U.S. Army Information System Software Center (ISSC) Software Development Center, Washington (SDC-W). This program has become the basis for the DSRS under the DISA CIM. Under this system, a repository has been established for each Service. The NWRC serves as the repository

for all Navy reusable components and provides an interface to the central repository and other service repositories under the DSRS.

Software Technology for Adaptable, Reliable Systems Repository
blanchard@stars.startech.com

STARS maintains a repository of Ada binding to Motif, Ada binding to Ada/Xt Windows Intrincints, Reuse Library Framework (RLF), and many more.

Tape copies are available from:
Asset Source for Software Engineering Technology (ASSET)
2611 Cranberry Square
Bldg. 2600, Suite 2
Morgantown, WV 26505
(304) 594-1762

MS-DOS high-density diskette copies are available from:
Advanced Software Technology, Inc.
P.O. Box 937
Medford, NY 11763
(516) 758-6545

A.1.6 Conferences and Special Interest Groups

ASEET Symposium
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria, VA 22311
(703) 825-6626

The ASEET Team Coordinator Working Group (CWG) sponsors the annual ASEET Symposium. The symposium enables DOD personnel to learn about new education and training methods from industry, academia, and DOD organizations.

DON Ada Users Group
DON Ada Users Group-Chair
Naval Ocean Systems Center
271 Catalina Boulevard
San Diego, CA 92152-5000
(619) 553-2303
FAX: (619) 553-5799

The DON Ada Users Group has been chartered to provide information and support to the DON on use of the Ada programming language and Ada-related issues associated with software development and maintenance. Regular meetings are held in conjunction with national conferences in the Ada community, such as those sponsored by Tri-Ada and SIGAda.

STARS Workshop
IDA/CSED
5111 Leesburg Pike
Falls Church, VA 22041
(703) 845-3520

The STARS Joint Program Office holds workshops to publicize and disseminate information on various contract efforts.

Software Technology Conference
Software Technology Support Center
Ogden ALC/TISE
Hill AFB, UT 84056
(801) 777-7703

The annual STC is held each April in Salt Lake City, Utah. This conference, sponsored by the STSC and the Headquarters of each Service, is a forum for sharing technology solutions and bringing together the DOD Government and contractor software community to exchange ideas and information.

A.1.7 Operational Development Support Tools

Ada Language System/Navy
Naval Sea Systems Command (PMS-412)
NC-3
2531 Jefferson Davis Highway
Washington, D.C. 20362-5101
(703) 602-8204

The ALS/N is a software development environment and Run-Time Environment (RTE) system that is being developed for the current generation of DON standard computers, the AN/UYK-43(V), AN/UYK-44(V), and AN/AYK-14(V). ALS/N has a users' group and a bulletin board available upon request and target-based training available through the project office.

AdaSAGE

Department of Energy
Idaho National Engineering Laboratory (INEL)
Idaho Falls, ID
(208) 526-0656
Internet: jjj@mica.inel.gov

AdaSAGE is a Government-owned development reuse tool utilized by all DOD components. The tool consists of utilities that support user-developed interfaces, reports, and data and their relationships. These utilities facilitate user-directed rapid prototyping. AdaSAGE is free and is supported by a Joint Service Configuration Management Board. Enhancements may be requested by leaving a message on the Ada Technical Support Bulletin Board listed in A.1.4.

NAVAIR Software Engineering Environment Tool Set

Charles Koch
Code 7033
Naval Air Development Center (NADC)
Warminster, PA 18974-5000
(215) 441-2752

The Naval Air Systems Command (NAVAIR) Software Engineering Environment (NASEE) Working Group contracted for 12 software tools for use throughout the software life cycle. NAVAIR has made these tools a standard for its Software Support Activities (SSAs). The NADC provides information on the NASEE Working Group and on ways to obtain these tools.

Tool Box PC

Software Technology Support Center
Hill AFB, UT 84056
1-800-477-2449

This tool, written in Ada, is an interactive catalog application tool that has Ada and other Government- and commercially owned software languages. This system is designed for managers' use. The program is available free to the public on 5¼- and 3½-inch disks. The Air Force supports this program through the STSC.

A.2 Ada INFORMATION CLEARINGHOUSE

The latest information about Ada is available free of charge from the AJPO's AdaIC. The AdaIC makes available information on a variety of topics ranging from the use of Ada within DOD and industry to tools and compilers for Ada developers, and from DOD policies regarding Ada to reusable Ada software.

The AJPO sponsors the AdaIC. The AJPO is responsible for informing the community about Ada, facilitating the language's implementation in the services, and maintaining the integrity of the language.

The telephone hot line numbers are 1-800-ADA-IC11 (232-4211) or (703) 685-1477. For answers to your Ada questions, call the AdaIC, Monday through Friday, from 8:00 a.m. to 5:00 p.m., Eastern Time.

Informational Flyers

More than 100 different informational flyers or reports are available from the AdaIC. Flyer topics include:

- Ada Validated Compilers
- Ada News and Current Events
- Ada Usage
- AJPO's Ada Technology Insertion Program (ATIP)
- Ada 9X project
- On-line sources of Ada information
- Ada bibliographies
- Ada Compiler Validation and Evaluation
- Resources for Ada Education and Training
- Ada Software, Tools, and Interfaces
- Ada Regulations, Policies, and Mandates
- Ada Historical Information
- Standards and Available Ada Bindings Products.

These flyers are available electronically on two AJPO-sponsored computer systems: the AJPO host computer on the Internet, and the AdaIC Bulletin Board. Paper copies of the flyers are provided upon request.

The Validated Compilers List is one of the most frequently requested flyers. This list, which is updated monthly, contains information on all compilers that have currently active validation certificates.

On-Line Information

Most AdaIC flyers and other publications are available on-line on the AJPO host computer and on the AdaIC Bulletin Board. These electronic sources also have other files, such as those whose length or complexity preclude easy distribution in paper-copy form. In addition, the AJPO provides information related to the Ada 9X project on a dedicated bulletin board and on the AJPO host.

The AJPO Host on the Internet

For those with access to the Internet, AJPO makes a variety of Ada information available on the AJPO host computer on the Internet. Its name is ajpo.sei.cmu.edu.

The AJPO host can be accessed by the File Transfer Program (FTP), which allows a user to transfer files to and from a remote network host site. FTP should work for any host on the Internet.

A sample FTP connection follows.

[your-prompt] ftp ajpo.sei.cmu.edu	execute ftp from your remote site
name: anonymous	login using "anonymous"
password: guest	enter password of "guest"
ftp> cd public	change to the "public" subdirectory
ftp> dir	view a list of accessible subdirectories
ftp> cd [sub-directory]	change to the subdirectory of your choice
ftp> dir	view a list of available files
ftp> get file1.hlp [newname.hlp]	get "file1.hlp" from ftp and copy it to "newname.hlp" on your machine
ftp> mget file1 ... fileN	get multiple files from ftp and load them onto your machine with the same names
ftp> bye	logout when finished

For more information on the AJPO host, type "get README" and "get README.FTP" after an ftp connection is made.

For those without Internet access, the AdaIC Bulletin Board is available on a 24-hour dial-up basis.

AdaIC Bulletin Board
Commercial: (703) 614-0215
AUTOVON: 224-0215

The AdaIC Bulletin Board contains most of the information provided on the AJPO host computer—plus on-line databases.

The bulletin board can be accessed by dialing one of the numbers listed above. Users should set their telecommunications package with the following parameters:

- Baud rate = 300 through 9600 baud
- Data bits = 8
- Parity = none
- Stop bits = 1.

The first time you log on, you will be prompted to register for an account.

The [D]oors feature of this bulletin board provides users with the capability to search six AdaIC databases:

- The Validated Ada Compilers (VCL) [D]oor provides full text searching of the Validated Compiler list by host and by target where different from host.

- The Ada Programming Tools (TOOLS) [D]oor contains information about more than 200 Ada vendors, their more than 300 Ada products, and the hardware on which they run.

- The Current Ada Articles (NEWS) [D]oor provides full text searching of AdaIC's abstracts of Ada-related articles that have been published in trade and technical journals.

- The Ada Bibliography (BIBS) [D]oor provides users with a comprehensive bibliography of Ada-related publications.

- The Bibliography/Abstracts (ABS-BIB) [D]oor provides users with a bibliography of Ada-related documents as well as an abstract for each bibliographic citation.

- The Ada Education (CREASE) [D]oor provides access to the AdaIC's "Catalog of Resources for Education in Ada and Software Engineering," Version 6.0, 1992.

Ada9X Bulletin Board
1-800-Ada-9X25 (1-800-323-9925)
(301) 459-8939

The Ada9X Project's electronic bulletin board is a comprehensive, one-stop source of information concerning the Ada9X project. All of the revision requests that were submitted to the project are available for viewing and/or downloading from the bulletin board. In addition, most of the project reports and all of the Ada9X project announcements are available.

The Ada9X Bulletin Board can be accessed by dialing one of the numbers listed above. Users should set their telecommunications package with the following parameters:

- Baud rate = 300 through 2400
- Data bits = 8
- Parity = none
- Stop bits = 1

Databases
In addition to the database available for searching on the AdaIC Bulletin Board, the AdaIC also maintains a database of Ada projects. The Ada Usage database was developed to track how Government, education, and industry are using Ada in software development efforts. Currently, there are more than 650 efforts described in the database.

Ada Usage information can be obtained only with the voluntary cooperation of the project. If you are currently involved in an Ada development project or if you have completed a project using Ada, we would like to add your information to our database.

Written Inquiries
If you prefer to send a written inquiry or would like to share any Ada-related information with us, send mail to:

AdaInformation Clearinghouse
P.O. Box 46593
Washington, D.C. 20050-6593

A.2.1 Public Access to the AdaIC Bulletin Board (ada-rbbs.hlp extract)

The AdaIC Bulletin Board is a publicly available source of information on the Ada language and Ada activities. Sponsored by the AJPO and maintained by AdaIC, this bulletin board is used to announce current events and general activities and provide a

current listing of validated Ada compilers. Access to the bulletin board requires a computer terminal and modem or a PC and modem.

The AdaIC Bulletin Board system can be accessed by dialing (703) 614-0215 or (301) 459-3865. Users should set their telecommunications package with the following parameters:

- Baud rate = 300, 1200, or 2400
- Parity = none
- Data bits = 8
- Stop bits = 1

Currently, the following 12 directories are available:

- The Ada Information Directory—an alphabetical listing of all available information files, with a contents description for each one
- The Language Reference Manual Directory—the Ada Language Reference Manual (ANSI/MIL-STD-1815A-1983) in its entirety
- The Approved Ada Commentaries Directory—approved commentaries responding to questions, problems, and/or inconsistencies and perceived inconsistencies regarding the Ada Language Reference Manual (ANSI/MIL-STD-1815A-1983)
- The Ada Language Rationale Directory—the rationale for the design of the Ada Programming Language in its entirety
- The CAIS Document Directory—CAIS documents (October 1986)
- The AdaIC Newsletter Directory—past AdaIC newsletters
- The CREASE Directory—AJPO'S *Catalog of Resources for Education in Ada and Software Engineering*, Version 5.0, in its entirety
- The Miscellaneous Directory—files such as those used to decompress compressed files
- Directories 9 and 10—a guidebook and reference manual, respectively, for the evaluation and validation of Ada programming support environments

- Directory 11—the NASA-Goddard Ada Style Guide, which was proposed as the basis for a military handbook

- Directory 12—a catalog of the ASR and the ASR User's Handbook.

Files are available in either compressed or uncompressed (ordinary ASCII text file) format. Most are available in both.

A.2.2 Access to Ada Information on the Defense Data Network (ada-ddn.hlp extract) The public directory on the *ajpo* host computer is an official source of information on the Ada language and Ada activities. Sponsored by AJPO and maintained by AdaIC, this computer directory is used to announce current events and general activities and to provide a current listing of validated Ada compilers.

This directory is available only to authorized users of the DDN. However, AdaIC also maintains a bulletin board at (703) 614-0215 and (301) 459-3865. For information, see the AdaIC handout, "Public Access to the Ada Information Bulletin Board" (AdaIC from G/V51, file ADA-RBBS.HLP).

The DDN is a collection of approximately 80 different computer networks representing DOD facilities, research centers, and academic institutions throughout the free world. All of the networks are packet-switching systems with interconnections at various locations. DOD controls access to the DDN. To obtain access to the DDN, it is first necessary to have an account or access to an account on one of the several thousand host computers that make up the system.

The following set of commands provides an example of the use of ftp to transfer a file from the *ajpo* host to a local host. The file is in the directory public/ada-info/val-comp.hlp.

```
>ftp ajpo.sei.cmu.edu
when asked for login name, type in anonymous
when asked for password, type in your user id
ftp>ls                    —provides listing of login directory
ftp> cd public            —changes directory to the public directory
ftp> cd ada-info          —changes directory to the ada-info directory
ftp> get val-comp.hlp     —copies file to your local directory
ftp> quit                 —returns control to UNIX
```

As of 17 March 1992, directories in the public directory include acvc-current, ada-adoption-hbk, ada-comment, ada-info, ada-lsn, ada-ui, ada9x, adanews, adastyle, artdata, asis, cais, crease50, ev-info, infoada, kitdata, lrm, pcis, piwg, rationale, and

wbs.sw. These files correspond to those shown in Table A-1. This appendix provides the primary Ada information files in the AdaIC File Directory.

A.2.3 Info_Ada Digest

DDN users can also access the Info_Ada Digest (to send discussions to the digest, use info_ada@ajpo.sei.cmu.edu). To request that you be added to the discussion list, use info_ada_requests@ajpo.sei.cmu.edu. Alternatively, the same discussions are available through USENEWS news group comp.lang.ada.

DDN users can also access the Ada_Ed Digest. To send discussions, use ada-ed@east.pima.edu. To request that you be added to the discussion list, use ada-ed-requests@east.pima.edu.

The Ada electronic mailing list includes the following:

- Ada announcements
- Open forum for discussion
- Open forum for questions
- Requests for information to the entire Ada community.

A.2.4 Document Reference Sources

In addition to the information available from AdaIC, many documents are available from sources described below. This information is taken from the AdaIC Document Reference List.

Government Printing Office

Superintendent of Documents
Government Printing Office
Washington, D.C. 20402
(202) 783-3238

The Government Printing Office (GPO) distributes the *Reference Manual for the Ada Programming Language* to the general public and industry for $16 a copy. Mail orders may be sent to the above address with payment included. Telephone orders are accepted with a VISA or Master Card number or a GPO deposit account number. For additional information, call the number noted above.

Government Source Codes

SEI= Software Engineering Institute
AJPO = Ada Joint Program Office
WPAFB = Wright Patterson Air Force Base

CECOM = Communications Electronic Command
USAF = United States Air Force

A.2.5 AdaIC File Directory

The information in Table A-1 was listed in the AdaIC Bulletin Board in March 1992. It details the types of information available from AdaIC.

Table A-1. AdaIC Directories

Directory Numbers and General Description of Contents*	
1 Ada Information Files	8 Miscellaneous— Unzipping Utilities
2 Language Reference Manual	9 APSE E&V Guidebook V2.0
3 Approved Ada Commentaries	10 APSE E&V Reference Manual
4 Ada Language Rationale	11 Proposed Ada Style Guide
5 CAIS Document	12 ASR User's Handbook & Directory
6 AdaIC Newsletter	
7 Catalog of Resources for Education (CREASE)	

* To list Directory, type l:1
For a list of all available files on the system, download director.zip

AdaIC Information Files—Directory 1

This directory contains electronic copies of the flyers and other documents offered by AdaIC. In addition, it contains electronic copies of several DOD directives relating to the Ada programming language.

The files below are listed with the extension ".HLP". When you use the download command, you will be prompted for the filename. If you give the filename with the .HLP extension, you will get an ordinary ASCII text file. However, to reduce the time required for downloading to your computer, most of the files listed below are also available in compressed (ZIPped) format. To download a file in compressed format, substitute .ZIP for the .HLP extension.

To view these ZIPPED files, you need an unzipping utility, which is available on this and many other bulletin boards. (See Directory 8 and Bulletin 2.)

File Name	Updated	Size	Contents
3405-1	7/18/89	18644	Text of 4/2/87 DOD Directive 3405.1, Computer Programming Language Policy
3405-2	7/10/89	7709	Text of 3/30/87 DoD Directive 3405.2 mandating use of Ada language in computers integral to weapon systems
9XDDN	7/09/91	6144	Access to Ada 9X information on DDN
9XNEWS	2/10/92	6144	Copy of the most recent Ada 9X Report to the Public
9XORDER	11/05/91	4096	How to order Ada 9X documents
ABSTRACT	12/20/91	20480	Abstracts of Ada-related articles
ACEC	8/15/91	61440	How to obtain the Ada Compiler Evaluation Capability (ACEC), DOD's compiler-performance test package
ACVC	4/29/91	40960	How to obtain a copy of the latest Ada Compiler Validation Capability (ACVC), the validation test suite
ADA-BIB	10/15/91	2048	How to obtain the AJPO'S Ada Bibliography, Volumes I, II, and III (1983–1986) and description
ADA-CALR	1/30/92	10240	List of upcoming conferences, symposia, and programs on Ada

ADA-DDN	8/06/91	6144	How to access ajpo.sei.cmu.edu, the *ajpo* host on the DDN
ADA-PROD	8/07/91	22528	List of articles and books on Ada costing, sizing, and productivity
ADA-RBBS	2/06/92	6144	How to access the AdaIC Bulletin Board at (703) 614-0215 or (301) 459-3865
ADA-USE	3/14/91	164839	Summary of the Ada Usage Database, which lists reported Ada projects from around the world
ADABOOKS	2/10/92	40960	Books relating to the Ada Programming Language
ADACPLUS	12/20/91	24576	Summary of Ada versus C++ Business Case Analysis Report
ADAIC	2/06/92	14336	A description of services offered by AdaIC
ADANET	3/15/91	4096	Text of AdaNet's Executive Summary describing its on-line services
ADATODAY	2/06/92	24576	On-line newsletter of current events and developments relating to Ada
ADAYEST	2/04/92	36864	Items archived from *Ada Today* (ADATODAY.HLP)
AEO-SEO	1/14/92	4096	Current list of Software Executive Officials (formerly AEO)
AF-IMP89	7/18/89	29081	Text of 1/1/89 Air Force Ada Implementation Plan

AF-INT91	8/12/91	2048	Text of Air Force 1991 Interpretation of Congressional Mandate
AF-POL88	11/09/88	41809	Text of 11/9/88 Air Force policy on programming languages
AF-POL90	12/21/90	10868	Text of 8/7/90 Air Force policy on programming languages
AI-ADA	8/12/91	24576	Ada and AI documents available from DTIC and NTIS
AJPO-891	10/28/91	6144	Article announcing that SPC's guide would be AJPO's suggested Ada style guide (with ordering information)
ARCHIVES	11/02/89	18341	Items archived from *Ada Yesterday* (ADAYEST.HLP) that are more than 1 year old
ARMIMP90	7/16/90	17928	Text of 7/16/90 Army Ada Implementation Plan
ASEETLIB	4/10/91	16446	Training-related materials in the ASEET Materials Library at the AdaIC
ATIP-F89	4/24/91	18432	Projects assisted by the Ada Technology Insertion Program in FY89
BENCHMRK	7/30/91	12288	How to obtain various benchmark performance test suites
BINDINGS	2/04/92	73728	Available Ada bindings
CLAS-SEM	2/06/92	51200	Classes and seminars relating to the Ada language

CREASE	11/27/91	2048	How to obtain AJPO's April 1988 CREASE Version 5.0
CREASFOR			Ada Education Survey form for CREASE Ver. 6.0
DEF-MCCR	3/04/83	4795	Text of 3/4/83 DOD guidelines for acquiring computer resources (defines mission-critical computer resources)
DOCU-REF	12/05/91	20480	List of Ada-related documents available through DTIC/NTIS and information on how to obtain them
DOORS	7/09/91	6144	AdaIC Databases Available, On-line Ada Products and Tools, and Ada Pragma Support
EMBDSYS	9/09/91	34816	Abstracts of documents and articles on Ada and embedded systems
FAA_ADA	11/07/89	6207	Text of 10/20/89 FAA Action Notice for mandating the use of Ada in acquisition and major modifications
GENINTRO	10/10/91	2048	Cover letter to accompany General Information Packet
GLOSSARY	8/11/90	47056	Ada-related terms and their meanings
GRAMMAR	10/04/89	37569	"A LALR(1) Grammar for ANSI Ada" by Gerry Fisher and Phillipe Charles, 1983
HISTADA	11/26/91	26624	"The History of Ada"—March 1984 article by Robert DaCosta

IMP[illegible]JIDE	11/26/91	2048	How to obtain the Ada Compiler Validation Capability Implementers' Guide (1986)
ISO-STAT	11/26/91	10240	Background information on the ISO's acceptance of Ada as an international standard
LADY-LOV	11/25/91	10240	Article on life of Ada Lovelace by Carol L. James and Duncan E. Morrill with note on the naming of the Ada language
LRM	11/26/91	4096	How to obtain the Ada Language Reference Manual, ANSI/ MIL-STD-1815A 1983
MAIL_DDN	8/20/91	51200	A list of UNIX public-access sites that can be used to send E-mail to hosts on the DDN
MANDAT90	1/28/92	6144	Text of the Congressional Ada mandate—plus some background
MARIMP88	3/09/88	32563	Text of 9 Jan 1988 Marine Corps Ada Implementation Plan
NATO-ADA	11/26/91	2048	Text of 1985 AJPO announcement of NATO's adoption of Ada as a common High Order Language (HOL) in military systems
NAVIPL91	11/26/91	20480	Interim Department of the Navy Policy on Ada, 24 Jun 1991
OODBIB	9/05/91	34816	List of articles and documents on Ada and Object-Oriented Design (OOD)
REALTIME	6/19/91	40960	List of publications on Ada used in real time jobs

REPOSTRY	11/26/91	14336	How to obtain programs and tools from the Ada Software Repository on SIMTEL20
REUSCODE	2/06/92	16384	Sources of Ada source code, reusable components, and software repositories
REUSEPUB	9/16/91	24576	List of publications relating to the reuse of Ada source code
SERIALS	11/26/91	12288	List of serial publications that feature information on the Ada language and the Ada community
STYL-ORD	11/05/91	2048	Ordering information and order form to order version 2 of Ada Quality and Style
SUCCESS	10/17/91	34816	Reprint of article from *Military & Aerospace Electronics*
TNG-TAPE	11/25/91	20480	Description and ordering information for a 19-tape series of Ada training videotapes
TRADEMRK	4/23/91	6144	Text of 1987 AJPO announcement that Ada trademark is replaced by certification mark
VAL-COMP	2/05/92	123280	List of the currently validated Ada compilers
VAL-DOC	7/03/91	2048	Instructions on how to obtain the Ada Compiler Validation Procedures
VAL-NOV	12/01/90	145846	List of validated Ada compilers as of Nov 90—kept for information purposes

VAL-PROC	9/19/90	55320	Text of the Ada Compiler Validation Procedures, Version 2.1, August 1990
VALCOVER	4/16/91	2048	Cover letter to accompany Validation packet
VALFACIL	12/04/91	2048	List of Ada Validation Facilities (AVFs) performing Ada Compiler Validation Capability tests
VSR-DOCU	7/03/91	24576	List of Validation Summary Reports (VSRs)—results from testing of compilers—and how to order info. from DTIC/NTIS
WITHDRWN	8/05/91	8192	Tests that have been withdrawn from the validation test suite, ACVC 1.11
X-SURVEY	11/01/91	12288	X/Ada binding user questionnaire of the X/Ada Study Team at GHG Corporation

HELPFUL GOVERNMENT SOURCES MATRIX

SOURCE	PAGE	STDS	TRNG	REFS	B-BRDS	PUBS	CONF.	TOOLS
Ada AND C++	A-17					X		
Ada SLICES	A-18					X		
Ada SOFTWARE REPOSITORY	A-26			X				X
Ada TECH B-BOARD	A-22				X			X
Ada VALIDATION OFFICE	A-3	X						X
Ada9X PROJECT	A-2	X			X			
AdaIC	A-3	X	X	X	X	X	X	X
AdaSAGE	A-10		X		X			X
AJPO	A-2	X	X	X	X	X	X	X
ALS/N	A-8		X		X			X
ASEET	A-9		X			X	X	
CAIS	A-11	X	X					X
CRSS	A-29			X				
DACS	A-19					X		X
DON Ada REPRESENTATIVE	A-4	X	X	X		X	X	X
DON Ada USERS GROUP	A-32	X	X	X	X		X	X
NASEE POC	A-34		X					X
NATIONAL AUDIOVISUAL CENTER	A-13		X					
NATIONAL TECH. INFO. SERVICE	A-16					X		
NAVCOMTELCOM Ada REP.	A-5		X	X	X			X
NAVY WIDE REUSE CENTER	A-31			X				X
RAPID	A-31			X				X
SEI	A-6		X			X	X	X
SPAWAR 224-1	A-4	X						
STARS	A-7			X		X	X	X
STNDS DOCS ORDER DESK	A-16	X				X		
U.S.ARMY ENG. COLLEGE	A-15		X					
USMC Ada REP.	A-5	X	X	X				

A.3 OTHER SOURCES

The information on commercial and nonprofit organizations cited below is provided to help the DON Program Manager become knowledgeable about Ada-related issues. These sources are not endorsed by the DON. They are provided to augment the list of Government sources in Section A.1 and to help Program Managers become familiar with the wide array of available sources.

Other sources (e.g., organizations, training, publications, tools) to be considered for inclusion in future editions of the Ada Implementation Guide should be sent to the following address:

Commander
Space and Naval Warfare Systems Command
SPAWAR 2241 (CDR M. Romeo)
2451 Crystal Drive
Washington, D.C. 20363-5100

A.3.1 Training

AdaWorks
261 Hamilton Avenue
Suite 320E
Palo Alto, CA 94301
Attn.: Richard Riehle
(415) 328-1815
FAX: (415) 328-1112
Internet: riehler@ajpo.sei.cmu.edu

AdaWorks trains DOD personnel in all aspects of Ada software development. Courses range from introductory through advanced Ada and include material tailored to the special needs of Management Information Systems (MIS) and/or COBOL programmers and analysts, scientific and embedded systems developers, and experienced software engineers. AdaWorks also provides Ada and software engineering training by giving project experience through a mentoring process.

Alsys
5959 Cornerstone Court West
San Diego, CA 92121
(619) 457-2700

Alsys has been a major Ada compiler vendor for the last 9 years. It has developed a set of training courses tailored to the installation and use of its compiler technology.

EVB Software Engineering, Inc.
5303 Spectrum Drive
Frederick, MD 21701
Attn.: Jennifer Jaynes Lott
(301) 695-6960
FAX: (301) 695-7734

EVB provides several courses including, but not limited to, Ada Programming, OOD and requirement analysis, XWindows, and software reuse in Ada and software development.

Fastrak Training Inc.
Quarry Park Place
9175 Guilford Road
Suite 300
Columbia, MD 21046-1802
(301) 924-0050

Fastrak presents both on-site and public courses in software engineering, object-oriented technology, and the Ada language.

Reifer Consultants Inc.
Marketing Manager
Reifer Consultants Inc.
P.O. Box 4046
Torrance, CA 90510
(310) 373-8728
FAX: (310) 373-9845

Reifer Consultants Inc., founded in August 1980, focuses primarily on consulting in Ada transition metrics, risk analysis, and cost estimating. They market a software sizing model and an Ada costing package. Training for these packages is provided through public and on-site seminars.

Texel Company
Victoria Plaza, Building 4, no. 9
615 Hope Road
Eaton, NJ 07724
(201) 992-0232

Texel specializes in Ada education and training consulting, Independent Validation and Verification (IV&V), and application development.

Universities and Colleges (Civilian)

The following universities and colleges are currently teaching Ada as the first language to their entering majors (Feldman, 92):

Allan Hancock College, California
Birmingham Southern College, Alabama
California State University, Long Beach, California
California State University, Northridge, California
California Polytechnic State University, San Luis Obispo, California
Cypress College, California
Embry-Riddle Aeronautical University, Florida
Florida Institute of Technology, Florida
Fayetteville State University, North Carolina
The George Washington University, Washington, D.C.
Indiana-Purdue University, Ft. Wayne, Indiana
LeMoyne College, New York
Marion County Technical Center, West Virginia
Marshall University, West Virginia
Muskingum College, Ohio
Norwich University, Vermont
Oklahoma City University, Oklahoma
Otterbein College, Ohio
Saint Mary College, Kansas
Sam Houston State University, Texas
San Diego Mesa College, California
Southern Arkansas University, Arkansas
State University of New York at Plattsburgh, New York
Stockton State College, New Jersey
University of Dayton, Ohio
University of New Orleans, Louisiana
University of South Dakota, South Dakota
University of South Florida, Florida
University of Washington, Washington
West Virginia University, West Virginia

The following universities and colleges first introduce Ada in their CS2 or Data Structures courses (Feldman, 92):

Briar Cliff College, Iowa
California Polytechnic State University, Pomona, California
California State University, Fullerton, California

College of West Virginia, Beckley, West Virginia
Daniel Webster College, New Hampshire
Florida International University, Florida
Gallaudet University, Washington, D.C.
Georgia State University, Georgia
Indiana University, New Albany, Indiana
Lenoir Rhyne College, North Carolina
Mesa State College, Colorado
Monterey Peninsula College, California
Murray State University, Kentucky
National University, California
Northern Arizona University, Arizona
Northern Kentucky University, Kentucky
Northeast Missouri University, Missouri
Oglethorpe University, Georgia
Ohio University, Athens, Ohio
Pennsylvania State University, Harrisburg, Pennsylvania
Portland State University, Oregon
Rose Hulman Institute of Technology, Indiana
Southwest Baptist College, Missouri
Shippensburg University, Pennsylvania
United States Air Force Academy, Colorado
University of Alaska, Fairbanks, Alaska
University of Missouri, Columbia, Missouri
University of Richmond, Virginia
University of Scranton, Pennsylvania
Western New England College, Massachusetts

A.3.2 Publications

AdaDATA Newsletter
International Resource Development, Inc.
P.O. Box 1716
New Canaan, CT 06840
(203) 966-2525

This monthly newsletter covers market trends and commercial developments in Ada software, services, and equipment. The cost of a subscription is $445 per year.

Ada Letters
Association for Computing Machinery, Inc.
1515 Broadway
New York, NY 10036
(212) 869-7440
Attn.: Membership Services
Internet: acmhelp@acmvm.bitnet

This bimonthly publication for the ACM SIGAda has been published since 1981. The newsletter contains technical Ada articles as well as a calendar of Ada events. (A subscription costs $20 per year for ACM members and $35 per year for nonmembers. Annual ACM membership dues are $79 for nonstudents and $24 for students. It costs $42 per year to become a SIGAda member only.)

Ada Newsletter
Raytheon Equipment Division
Tim Boutin, Editor
MS 5-2-508
Sudbury, MA 01776
(508) 440-3607

This newsletter tracks developments in the Ada language through conference reports and provides vendor news articles and a listing of Ada events. There is no charge for this publication.

Ada Rendezvous
Texas Instruments Incorporated
David G. Struble
Software Engineering Department
MS 8489
P.O. Box 869305
Plano, TX 75086
(214) 575-5346

Ada Rendezvous is a free annual publication. Articles span multiple areas of interest, including results of Ada compiler evaluations for embedded targets, review of Ada tools, and technical information contributed by Ada developers. Such articles provide guidance to application programmers on how to use Ada with specific hardware architectures and microprocessor designs. *Ada Rendezvous* also addresses evolving Government and DOD issues that affect existing and proposed contracts with Ada requirements.

Ada Strategies
Ralph E. Crafts, Editor and Publisher
Route 2, Box 713
Harpers Ferry, WV 25425
(304) 725-6542

This monthly newsletter covers Ada business strategies and contract-evaluation guidelines. It provides information on Ada policy and trends and on Congressional and funding issues as well as insight concerning current legislation. The annual cost is $100 for Government subscribers.

CAUWG Report
Alsys, Inc.
67 South Bedford Street
Burlington, MA 01803-5152
(617) 270-0030

This newsletter for members of ACM SIGAda's Commercial Ada Users Working Group (CAUWG) contains news and comments. It is available to the public at no charge.

FRAWG Newsletter
Martin Marietta Aerospace
MS L0420
P.O. Box 179
Denver, CO 80201
(303) 971-6731

This newsletter is a publication of the Front Range Ada Working Group (FRAWG). There is no charge for this publication.

Software Engineering Notes
Association of Computing Machinery, Inc.
1515 Broadway
New York City, NY 10036
(212) 869-7440
acmhelp@acmvm.bitnet

This quarterly is an informal publication of the ACM Special Interest Group on Software Engineering (SIGSOFT), which is concerned with the design and development of high-quality software. (A subscription costs $16 per year for ACM members and $38 for affiliate nonmembers. Annual ACM membership dues are $75 for nonstudents and $22 for students.)

SPC Quarterly
SPC Building
2214 Rock Hill Road
Herndon, VA 22070
(703) 742-8877

The *SPC Quarterly* is published by the Software Productivity Consortium (SPC) for unlimited distribution to its member companies, as well as to commercial, government, and academic organizations. SPC helps its member companies to develop the processes, methods, tools, and services needed to significantly improve the design and implementation of high-quality, software-intensive systems. Its methods seek to make the Ada software developer more productive. Use of these methods helps bridge the gap between well-established software engineering principles and the actual practice of programming in Ada. There is no charge for this publication.

A.3.3 Repositories

COSMIC, University of Georgia
382 East Broad Street
Athens, GA 30602
(706) 542-3265
FAX: (706) 542-4807

COSMIC distributes NASA-developed software including string, numerical, service, and linear algebra subprograms. Many are oriented to avionics applications. Source code is provided with the software purchase, and a free brochure is available.

EVB Software Engineering, Inc.
5303 Spectrum Drive
Frederick, MD 21701
1-800-877-1815
(301) 695-6969
FAX: (301) 695-7734

Generic Reusable Ada Components for Engineering (GRACE™) is a library of 275 Ada software components based on commonly used data structures such as strings, trees, and

graphs. Each component includes complete design documentation, source code, and at least one test program. GRACE is completely portable. Its only requirement is a validated Ada compiler. Free samples are available.

IWG Corp.
1940 Fifth Avenue
Suite 200
San Diego, CA 92101
(619) 531-0092
FAX: (619) 531-0095

Proplink is an Ada program for analysis of communication link propagation paths from Extremely Low Frequency (ELF) to Extremely High Frequency (EHF) using fast-running models.

MassTech, Inc.
3108 Hillsboro Road
Huntsville, AL 35805
(205) 539-8360
FAX: (205) 533-6730

Math Pack contains over 320 Ada mathematical subprograms in 19 reusable generic Ada packages. It includes linear algebra, linear system solutions, integration, differential equations, eigensystems, interpolation, probability functions, Fourier transforms, and transcendental functions. Purchase includes source code, documentation, on-line help, and telephone support.

Rockwell International Corporation
Manager, Software Engineering Process Group
M/S 460-220
3200 East Renner Road
Richardson, TX 75082-2402
(214) 705-0000

Rockwell International Corporation maintains a database server that contains the Ada tools set. It also maintains two libraries. One contains the implementor's tools and the other is a library of implemented software.

Wizard Software
2171 South Parfet Court
Lakewood, CO 80227
(303) 986-2405

Booch components feature data types and tools for sorting, searching, and character matching. Each abstraction has multiple implementations and follows OOD. Source code is provided. A version in C++ is also available. These products are also marketed in Europe and Japan.

A.3.4 Conferences and Special Interest Groups

SIGAda

Mr. Mark Gerhardt
ESL, Inc. MS G1
495 Java Drive
Sunnyvale, CA 94088-3510
(408) 752-2459
(408) 738-2888 (switchboard)

SIGAda is a professional society dedicated to the dissemination of information about all aspects of the Ada programming language, including standardization, implementation, usage, policy, management, and education. It sponsors meetings several times a year and also publishes a bimonthly newsletter, *Ada Letters*. Originally known as AdaTEC, SIGAda was established under the auspices of ACM in 1981. In addition to the national SIGAda organization, there are approximately 50 chartered local SIGAda chapters. Most of these local chapters hold technical meetings on a monthly basis. The point of contact for each local chapter is published in *Ada Letters*. The Washington, D.C., chapter of SIGAda holds an annual symposium on Ada.

Tri-Ada Conference

Danieli & O'Keefe Associates, Inc.
Chiswick Park
490 Boston Post Road
Sudbury, MA 01776
(508) 443-3330
1-800-833-7555 (in the United States and Canada only)
FAX: (508) 443-4715

Tri-Ada, SIGAda's major annual conference and exposition, combines the availability of lectures about the technology and management of the latest developments in the Ada community with in-depth presentations on project experience. The conference offers tutorials, birds-of-a-feather sessions, and the opportunity to see the Ada products and services available in the marketplace. In addition, information gathered in the coffee klatches and informal gatherings, which always occur at these meetings, is not obtainable

in any other way. Tri-Ada presents a unique opportunity to be immersed in the happenings in the world of Ada so that organizations can become or continue to be at the forefront of Ada understanding and use.

Washington Ada Symposium
Washington, D.C. SIGAda
(301) 286-7631
Ed Seidewitz

At the Washington Ada Symposium (WAdaS), information is presented on software engineering dealing with commercial industry, Government, military, scientific, academia, weapons, and administration with regard to Ada.

A.3.5 Operational Development Support Tools

ObjectMaker
Mark V Systems Limited
16400 Ventura Boulevard, Suite 303
Encino, CA 91436
(818) 995-7671

ObjectMaker (formerly Adagen) is a CASE tool that supports object-oriented diagramming methods for requirements analysis and top-level and detailed design. User-interface and diagram types are tailorable. Optional language modules automatically generate compilable code (C, C++, or Ada) from detailed design diagrams and support reverse engineering. The language module reverse engineering toolset takes legal code (C, C++, or Ada) back to multiple-level, nested, and detailed design-level diagrams. This is powerful for reuse and reengineering as well as documenting as-built code and component libraries. Older methods supported include data flow diagrams, real-time extensions, entity relationship, state transition, and structure charts. ObjectMaker is available on Digital Equipment Corporation, Sun, Apollo, Hewlett-Packard, MIPS, and Data General Aviion workstations as well as on Macintoshes and IBM PCs.

EVB Software Engineering, Inc.
5303 Spectrum Drive
Frederick, MD 21701
Attn.: Jennifer Jaynes Lott
(301) 695-6960
FAX: (301) 695-7734

GRAMMI is an Ada user interface toolkit that supports the development of Ada Graphical User Interfaces (GUIs) using the XWindows system. GRAMMI supports the

rapid prototyping and evolutionary development of Ada user interface software with an integrated set of tools that helps users to interactively build screens and generate the resulting Ada code. GRAMMI User Interfaces are designed to support the full features of Ada programs, including Ada tasking and exception handling.

HERAGRAPH is a two- or three-dimensional graphics application framework that enables the development of high-performance, interactive graphics applications. Use of HERAGRAPH's reusable graphical objects, Motif style user interface components, and application framework frees users to concentrate on developing the functionality of their applications. Written in Ada, HERAGRAPH provides an industrial-strength solution for the development of today's modern and complex graphics applications on both DOS and UNIX/XWindows platforms. HERAGRAPH has been used successfully in a variety of interactive, high-performance graphics applications. Domains include Geographic Information Systems (GISs), Range Control Applications, Railway and Transportation Control systems, Interactive Graphical Database Editor applications, Graphical Simulation Systems, and Graphical Training Systems.

REUSE LIBRARY TOOLSET (RLT) is an integrated set of tools that supports the definition, population, and searching of a software reuse library. Software components in RLT are classified using a faceted/attribute classification schema. Defining and populating a reuse library with RLT is performed entirely through a point-and-click GUI. The search and retrieval screens are automatically generated. There are no cryptic commands to learn or files to edit. RLT can be used to maintain large repositories of reusable software, yet it is easy enough for individual engineers to organize their own personalized libraries.

EVB Object-Oriented Development Method/CASE Tool Support. This tool is supported by Paradigm Plus by ProtoSoft. Paradigm Plus is a configurable CASE tool that uses the object-oriented model to provide support to a wide range of software engineering activities throughout the software life cycle. The tool is in use in more than 200 installations world wide. On the EVB edition of Paradigm Plus, EVB provides all customer support for the product. Questions about methodologies are answered directly by the source, assuring you of accurate answers at all times. This tool provides all the graphical notations and rules necessary to develop Ada systems using the EVB object-oriented approach.

Appendix B

Department of the Navy Standards, Policies and Procedures

This appendix provides a list of DOD/DON software policies in the following categories:

DOD Directives, Instructions, and Standards
DON Policies including:
- Secretary of the Navy Instructions
- Naval Operations Instructions
- Marine Corps Orders
- Naval Air Systems Command Instructions
- Naval Sea Systems Command Instructions
- Space and Naval Warfare Systems Command Instructions
- AV Documents
- Military Standards

DOD/DON SOFTWARE POLICIES

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
DODD 5000.1	Defense Acquisition	23 Feb 91	This directive provides policies, procedures, definitions, and responsibility descriptions relative to managing major and non-major defense acquisition programs, except when statutory requirements override. Policies governing streamlined acquisition organization structures, acquisition phases, and milestone decision points; affordability assessments; tailored acquisition strategy, are described. Procedures relative to the Defense Acquisition Board (DAB) and DAB Committee reviews are also addressed.	
DODI 5000.2	Defense Acquisition Management Policies and Procedures	23 Feb 91	This instruction provides supplemental guidance to DODD 5000.1 through detailed discussion of the milestone decision points, pre-DAB activities, and required program documentation per milestone. DOD 5000.2-M identifies and provides formats for requirements documents, acquisition documents, and independent documents associated with the acquisition process.	DODD 5000.1

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
DOD-STD-2167A	Military Standard Defense System Software Development	29 Feb 88	This standard establishes requirements to be applied during acquisition, development, and support of software systems. It also offers guidance relative to the development of Computer Software Configuration Items (CSCIs), contractual Statements of Work (SOWS), and the Contract Data Requirements List (CDRL). For total system development, this standard is intended for use in conjunction with MIL-STD-499A, Engineering Management. The following requirements are also described: software development management and engineering, formal qualification testing, software product evaluations, software configuration management, and transition support. More specific topics are also addressed: system requirements analysis and design, software requirements analysis, preliminary/detailed design activities, coding and Computer Software Unit (CSU) testing, Computer	DODI 5000.2 See MIL-STD 498 draft.
DOD-STD-2168	Military Standard Defense System Software Quality Program	29 Apr 88	This standard establishes requirements for a software quality program during application acquisition/development and support of software systems. This guidance applies to contract clause items, the Statement of Work (SOW), and the Contract Data Requirements List (CDRL).	DODI 5000.2
DODD 7920.1	Life-Cycle Management of Automated Information Systems (AISs)	20 Jun 88	No longer applicable.	Replaced by DODD 8120.1.

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
DODI 7920.2	Automated Information Systems (AISs) Life Cycle Management Review and Milestone Approval Procedures	7 Mar 90	No longer applicable.	Replaced by DODD 8120.2.
DOD-STD-7935A	Military Standard DOD Automated Information Systems (AISs) Documentation Standards	31 Oct 88	This standard provides guidelines for the development and revision of Life-Cycle Management (LCM) documentation for AI. Specific content and format requirements are described for the following documents: Functional Description (FD); specifications for the System/Subsystem, Software and Data Base; User/End User, Computer Operations and Maintenance manuals; Test Plan; Test Analysis Report; and Implementation procedures.	See MIL-STD 498 draft.
DODI 8000.1	Defense Information Management Program	27 Oct 92	This instruction establishes the Defense Information Management Program. It provides top-level policy and describes responsibilities for the management of Information Systems (ISs) under the cognizance of ASD (C3I).	

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTIONS/COMMENTS
DODI 3405.1	Computer Programming Language Policy	2 Apr 87	This instruction establishes ADA as the single DOD High Order Language (HOL). The common computer programming language for DOD.	DODD 8000.1
DODD 8120.1	Life-Cycle Management of Automated Information Systems	14 Jan 93	This directive updates policy, responsibilities, and procedures for DOD LCM of the AISs.	DODD 8000.1 Replaces DODD 7920.1.
DODI 8120.2	Automated Information Systems Life-Cycle Management	14 Jan 93	This instruction establishes procedures for the review and milestone approval for AIS programs as defined in and subject to DODD 8120.1.	DODD 8000.1 DODD 8120.1

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTIONS/COMMENTS
DODI 8020.1 Draft	Functional Process Improvement		This instruction designates the use of the Integrated System Definition Language (IDEF) automated tool for process analysis and defines the required function economic analysis required to support improvement decisions.	DODD 8000.1
SECNAVINST 4200.32	Design to Cost	12 Jul 84	No longer applicable.	Cancelled by SECNAVINST 5000.2A.
SECNAVINST 4210.6A	Acquisition Policy	13 Apr 88	No longer applicable	Cancelled by SECNAVINST 5000.2A.

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
SECNAVINST 4210.7A	Effective Acquisition of Navy Material	16 Jan 87	This instruciton establishes policies and assigns responsibilities to promote effective material acquisition through the use of nondevelopmental items (NDIs) to fulfill Navy requirements.	Cancelled by SECNAVINST 5000.2A.
SECNAVINST 4210.9	Acquisition and Management of Technical Data and Computer Software	25 Jan 88	No longer applicable.	Cancelled by SECNAVINST 5000.2A.
SECNAVINST 4855.1	Quality Assurance Program	10 Sep 79	No longer applicable.	Cancelled by SECNAVINST 5000.2A.

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTIONS/COMMENTS
SECNAVINST 4858.2E	DON Value Engineering Program	6 Jul 84	No longer applicable.	Cancelled by SECNAVINST 5000.2A.
SECNAVINST 5000.1C	Major and Non-Major Acquisition Programs	16 Sep 88	No longer applicable.	Cancelled by SECNAVINST 5000.2A.
SECNAVINST 5000.2A	Major and Non-Major Acquisition Program Procedures	9 Dec 92	This instruction implements DON procedures for defense acquisition.	DODI 5000.2

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
SECNAVINST 5000.39A	Acquisition and Management of Integrated Logistics Support (ILS) for Systems and Equipment	3 Mar 86	No longer applicable.	Cancelled by SECNAVINST 5000.2A.
SECNAVINST 5200.32A	Acquisition Management Policies and Procedures for Computer Resources	3 May 93	This instruction provides DOD acquisition policy for the selection of computer resources based on an open systems approach. The policy emphasizes the use of commercial-based computer resource products interface standards. A computer resource management cross-index to DODI 5000.2 and DOD 5000.2-M is included.	DODD 5000.1 DODI 5000.2 SECNAVINST 5000.2A DODD 8120.1 DODI 8120.2 SECNAVINST 5231.1C DOD-STD-2167A DOD-STD-2168 SECNAVINST 5239.2 SECNAVINST 5420.188C MIL-STD-973 MIL-STD-188 Series MIL-STD-882 DOD-STD-881 MIL-STD-499 MIL-STD-2036
SECNAVINST 5230.9A	Information Resources Program Planning	16 Oct 85	This instruction provides policies and responsibilities for program planning. A revision of it is in progress.	SECNAVINST 5230.4 SECNAVINST 5231.1B SECNAVINST 5224.1

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
SECNAVINST 5232.1	Quality Assurance (QA) Program for Information System (IS) Projects	16 Oct 85	This instruction stablishes a QA program for IS projects in DON.	SECNAVINST 5231.1C
SECNAVINST 5233.1B	DOD Automated Data Systems Documentation Standards	25 Jan 79	This instruction discusses DON implementation of the DoD standard and contains instructions for the preparation of documents.	SECNAVINST 5200.28 SECNAVINST 3560.1
SECNAVINST 5234.2	Computer Programming Language Policy	3 Nov 88	This instruction implements DOD Directive 3405.1 and DOD Directive 3405.2. It specifies that software be acquired based on analysis of life-cycle costs and impact, using the following order of preference: (1) nondevelopmental item (NDI) software, (2) Ada-based software tools, and (3) approved standard HOLs, mandates use of Ada for mission-critical systems (with some exceptions) and require Adas for all other applications unless use of another approved HOL (listed in an enclosure) is more cost-effective over the life cycle of the application. Achieving the long range goal of transition to Ada requires validation of Ada compilers and use of an/and Ada-based program design language and modern software engineering principles. Establishes responsibilities and waiver the process.	DODD 3405.1 DODD 3405.2 SECNAVINST 5000.1B SECNAVINST 5200.32 SECNAVINST 5231.1B DOD-STD-2167A SECNAVINST 5233.1B DODD 5010.19 SECNAVINST 4130.2

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
SECNAVINST 5236.1B	Contracting for ADP Resources	15 Oct 80	This instruction revises policies relating to contracting for ADP resources in DON.	SECNAVINST 5231.1C
SECNAVINST 5238.1C	Computer Resources Management	7 Apr 89	This instruction establishes policies and procedures for inventory management, sharing, and reutilization or redistribution of computer resources within DON.	DODD 7950.1
SECNAVINST 5239.2	Department of the Navy Automated Information Systems	15 Nov 89	This instruction establishes the DON AIS security program, sets forth policies and guidelines for and defines its organizational responsibilities for executing the program elements. *A draft SECNAVINST 5239.2A is in Staffing.*	DODI 5215.2 DODD 5200.28

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTIONS/COMMENTS
SECNAVINST 5420.188C	Navy and Marine Corps Program Decision Meetings	16 Jul 92	This instruction provides guidance regarding streamlining of the acquisition review process and provides procedures for conducting Navy and Marine Corps Program Decision Meetings.	DODD 5000.1 DODI 5000.2 SECNAV 5000.2A SECNAVINST 5231.1C
SECNAVINST 5230.11	Information Resources Management Review Program	19 Oct 89	This instruction establishes DON IRM Review Program.	DODI 7740.3
SECNAVINST 4130.2	DON Configuration Management Policy	11 May 87	No longer applicable	Cancelled by SECNAVINST 5000.2A.

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
SECNAVINST 5231.1C	Life-Cycle Management Policy and Approval Requirements for Information System Projects	10 Jul 92	This instruction updates the policy relative to LCM as the standard discipline for managing and obtaining approval for IS projects. Arevision in progress Cancels 5231.B 5236.5 3 SECNAV Notes: 5236 dated 16 Oct 85 5231 dated 8 Nov 88 5231 dated 2 Jun 89	DODD 8120.1 DODI 8120.2
SECNAVNOTE 5200	Acquisition Management Policies and Procedures for Computer Resources	20 May 93	This note provides the Open Systems Interface Standards List (OSISL) and the products accepted list as guidance to support computer resource selections under SECNAVINST 5200.32A.	SECNAVINST 5200.32A
OPNAVINST 3960.10C	Test and Evaluation	14 Sep 87	No longer applicable	Cancelled by OPNAVINST 5000.42D

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
OPNAVINST 4790.2B	Naval Aviation Maintenance Program	26 May 82	This instruction is the basic document and authority governing management of all Naval Aviation Maintenance. It/the affects handling of Fleet hardware that is part of the weapon system and supporting items. Maintenance procedures for equipment used in the operating system or its support are updated. Command, administrative, and management relationships are established, with policies and procedures for the assignment of maintenance tasks and/or responsibilities for conduct of the Program.	
OPNAVINST 5000.42D	Research, Development, and Acquisition Procedures OPNAV Role and Responsibilities in the Acquisition Process	19 Apr 93	This instruciton amplifies the general guidance in DOD Directive 5000.1. This instruction has largely been superseded by DOD 5000.2. The program initiation portions remain in effect and are being refined in an upcoming revision. DOD "5000" addresses Navy-unique implementation procedures for the series and SECNAVINST 5000.2A. Provides guidance on Acquisition Review Boards (ARBs) roles; requirements documentation, Cost and Operational Effectiveness Analysis (COEA), Integrated Program Summaries (IPSs), Acquisition Program Baseline (APB); and Test and Evaluation.	DODD 5000.1 DODI 5000.2 SECNAVINST 5000.2A
OPNAVINST 5200.28	Life-Cycle Management of Mission-Critical Computer Resources for Navy Systems Managed Under the Research, Development, and Acquisition Process	24 Sep 86		DODD 5000.1 DODI 5000.2 Cancelled by OPNAVINST 5000.42D.

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
OPNAVINST 5239.1A	Department of the Navy Automatic Data Processing Security Program	3 Aug 82	This instruction establishes the DON ADP Security Program for all ADP activities and networks and assigns responsibilities. Enclosures contain executive briefs on ADP Security and the DON Security Manual.	DODD 5200.28 DODD 5215.1 SECNAVINST 5239.2
OPNAVINST 9410.5	Database and Communication Standards Interoperability Requirements for Tactical Naval Warfare Systems	24 May 90	This instruction provides a management framework for Navy to resolve interoperability issues that adversely affect tactical naval warfare.	DODD 4630.8
OPNAVINST 5000.19B	Integrated Logistics Support in the Acquisition Process	10 Jan 87	This instruction establishes policy and procedures for establishing an Integrated Logistics Support (ILS) program for many systems. This includes establishing a computer resources program. It also contains requirements for Computer Resources Life-Cycle Management Plan (CRCCMP) review and approval.	

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
OPNAVINST 4105.3	Integrated Logistics Support Review and Appraisal	16 Jul 86	This instruction establishes policies and procedures for conducting LRG audits on Navy systems. One of its elements is Computer Resources Support.	
OPNAVINST 5000.49B	Integrated Logistics Support in the Acquisition Process	10 Jan 87	This instruction establishes the policy and procedures for sitting up an ILS program for many systems. This process includes establishing a computer resources program. This instruction also contains requirements for Computer Resources Life-Cycle Management Plan (CRLCMP) review and approval.	
Marine Corps Bulletin 3900	Tactical Software Management and Funding	7 Jan 88	This bulletin publishes new policy decisions regarding the management and funding of software support for tactical systems.	

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
MCO 4130.2	Configuration Management	17 Oct 77	This Marine Corps Order (MCO) prescribes policies and procedures and assigns responsibilities for configuration management support of computer programs and equipment of designated field tactical systems and their training support systems.	
MCO P4130.8	Configuration Management Manual	4 Jan 89	This order provides the policy and procedures required to implement configuration management within the Marine Corps and to provide a systematic methodology for documenting and controlling the configuration of material and equipment.	
MCO 5200.23A	Management of Mission-Critical Computer Resources in the Marine Corps	30 Dec 86	This order implements *DODD 3405.1&2, 5000.29,* SECNAVINST 5200.32 & 5234.2. It establishes policy for the acquisition, management, and life-cycle support of Mission-Critical Computer Resources (MCCR) in the Marine Corps.	DODD 3405.1 DODD 3405.2 DODD 5000.29 SECNAVINST 5200.32 SECNAVINST 5234.2

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
MCO 5230.15	Data Base Administration	9 Aug 83	This order establishes the organization and responsibilities for data administration as it applies to the design, development, implementation, and management of databases within the Marine Corps.	
MCO P5231.1	Life-Cycle Management for Information Systems Projects	17 Sep 87	This order establishes Marine Corps policy and regulations governing the development, implementation, operation, and management of information systems projects.	
MCO 5234.4	Configuration Management for ADP System Software	31 May 84	This order establishes standard procedures for planning, justifying, testing, evaluating, acquiring, and implementing ADP systems software in the Marine Corps.	

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTIONS/COMMENTS
MCO 5271.1	Information Resources Management Standards and Guidelines Program	19 Sep 86	This order establishes the Information Resources Management (IRM) Standards and Guidelines Program. It also guides the implementation of Marine Corps IRM policy by the development and distribution of publications that set technical standards for the management of all IRM activities.	
MCO 5271.2	Information Resources Management Program Planning	29 Sep 86	This order establishes policy and objectives and assigns responsibilities for IRM Program planning.	
MCO 5271.3	Management Oversight of Information Systems	16 Mar 88	This order assigns responsibilities for management oversight of information systems and supporting information resources. The assigned responsibilities address systems development, system operation, and the use of hardware and system software in support of developing and operational information systems.	

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
MCO P5510.14	ADP Security Manual	2 Jan 81	This order establishes policy and procedures for ADP security by addressing physical security, communications, emanations, hardware, software, procedural, risk management, contingency planning, and other security aspects contributing to the protection of an ADP system, site, facility, or operation.	
NAVAIRINST 3960.2B	Test and Evaluation	30 May 89	This NAVAIR instruction supports OPNAVINST 3960.10 for implementing and for assigning responsibilities within NAVAIR as a function of Project scope. The rates of the Advanced Development Project Officer, Project Manager, and/or Acquisition Manager are defined and identified as are their overall project responsibilities. AIR-05 and AIR-06 are designated as sharing the functional responsibility for Test and Evaluation. Details are provided as to the role in the review and approval of test plans by the NAVAIR structure.	OPNAVINST 3960.10
NAVAIRINST 4000.14A	Navy-Prepared Integrated Logistics Support Plans and Operational Logistics Support Plans for Aeronautical Systems and Equipment	3 Jun 75	This instruction promulgates the procedures and requirements for the development and preparation of the logistics support plans and equipment to be procured by or for the Naval Air Systems Command HQ. The enclosures provide instructions and formats for preparation or distribution of ILS Requirements Outline, ILS Plans, and Operational Logistics Support Plans.	SECNAVINST 5000.2A

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
NAVAIRINST 4130.1C	NAVAIR Configuration Management Manual	31 Jan 92	This instruction provides a comprehensive background of the configuration management process, including requirements for procedures to evaluate, implement, and record configuration changes for NAVAIR hardware and software. It contains flowcharts to illustrate the processes and contains sample forms as guides. It covers the life cycle of hardware configuration items from concept definition through full-scale development and the operational life of the system. Software configuration management procedures are similarly covered from the Software Configuration Management Plan in concept development, through the final product baseline and test reports. The instruction defines the Software Configuration Control process through system development and initial deployment. NAVAIRINST 5230.6 addresses configuration management during the post-deployment software support	NAVAIRINST 5230.6 MIL-STD-973
NAVAIRINST 4200.14B	Policy and Guidelines for Procurement of Data and Specific Acquisition of Unlimited Rights in Technical Data	1 May 79	This instruction stablishes policy and provides procedures for the acquisition of unlimited rights in technical data by NAVAIR and Field Agencies. It covers hardware, software, and process data. It is intended for use with Defense Acquisition Regulations (DARS) and MIL-STD-1679. DAR 9-601 which provides definitions and DAR 7-104.9(a) which addresses the use of "limited rights in software" markings should be consulted.	MIL-STD-167? DAR 9-601 DAR 7-104.9(a)
NAVAIRINST 4275.3F	Implementation of Configuration Control for DOD-STD-480A and MIL-STD-481A	17 Sep 85	This instruction provides specific guidance for selecting and implementing the Configuration Control Requirements of DOD-STD-480A and MIL-STD-481A. It defines the difference between the two on the basis of responsibility for an entire system (DOD-STD-481A). Specific actions required by the inclusion of either standard within a contract structure are outlined in detail. It affects configuration management procedures and scope of change analysis at either component or system level.	DOD-STD-480A MIL-STD-481A

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTIONS/COMMENTS
NAVAIRINST 5100.3C	Naval Air Systems Command System Safety Program	10 Dec 92	This instruction establishes policy and assigns responsibilities for a formal system safety process within NAVAIR. The instruction applies programs for which NAVAIR provides system safety engineering support for systems, subsystems, equipment, and facilities. The system safety program directs attention to early recognition of hazards and their elimination or control.	DODINST 5000.2 SECNAVINST 5000.2A MIL-STD-882B
NAVAIRINST 5215.12	NAVAIR Technical Directive System	16 Jan 90	This instruction describes the policy and procedures governing the NAVAIR HQ Technical Directive (TD) System. The TD System is the authorized medium for directing the accomplishment and recording of modifications and one-time inspections of NAVAIR-accepted equipment. The software component of fielded NAVAIR systems may be the subject of a TD bulletin. Changes to the software in fielded NAVAIR systems will be documented by a TD change.	SECNAVINST 5000.2A
NAVAIRINST 5230.5	Responsibility and Requirements for Preparation of Software Life-Cycle Management Plans	21 Jul 76	This instruction identifies activities responsible for the preparation and maintenance of Software Life-Cycle Management Plans (SLCMPs). An enclosure provides detailed instructions for the format and content of SLCMPs. The SLCMP addresses the operational software requirements for the complete life cycle of the weapon system. It is mandatory that the requirements described in the SLCMP be included in any Request for Proposals (RFP) issued for full-scale development. Upon approval by the Project or Acquisition Manager, the SLCMP shall become the governing document for operational software life-cycle support.	SECNAVINST 5000.2A

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTIONS/COMMENTS
NAVAIRINST 5230.6	Establishment of Mission-Critical Computer Resources Software Change Review Boards Naval Air Systems Command's	14 Jun 83	This instruction provides for the establishment of Software Change Review Boards (SCRBs). Enclosures provide an SCRB charter and a list of SCRB guidance documents. This instruction directs Project or Acquisition Managers to establish SCRBs during the validation or full-scale development phases when the project or acquisition manager judges such a formal review structure to be in (NAVAIR's) interests. It directs the creation of an SCRB before Fleet introduction unless all affected organizations and activities agree it is unnecessary.	
NAVAIRINST 5230.9	Policy and Procedures for the Establishment and Operation of Naval Air Systems Command Systems Software Support Activities	14 Jun 83	This instruction establishes the requirements for software support activities and promulgates policy, procedures, responsibilities, and operating relationships pertaining to their missions, functions, directions, and support. It specifically designates Navy activities to be assigned software support by warfare/functional category.	SECNAVINST 5000.2A
NAVAIRINST 5230.11 (Draft)	Policy and Procedures for the Development of Mission-Critical Computer Resources as Part of Naval Air Systems Command Weapon System/Equipment		This instruction provides policy, procedures, and assigns responsibility for the development of MCCR as part of weapon system/equipment procurements. It establishes requirements for Computer Resources Working Groups (CRWGs) and Computer Resources Life-Cycle Management Plans (CRLCMP). The CRLCMP replaces the SLCMP previously required by NAVAIRINST 5230.5.	NAVAIRINST 5230.5 SECNAVINST 5000.2A

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTIONS/COMMENTS
NAVAIRINST 5400.14C	Cognizant Field Activity Program	27 Dec 82	This instruction establishes and explains the procedures by which items of Naval Air Systems Command (NAVAIR) equipment being developed are put under the cognizance of designated field activities. It explains the responsibilities of these agencies and NAVAIR in the procurement of Service-approved equipment through its life cycle of support. Various items of equipment constituting a complete weapons system must be supported through the appropriate cognizant agency. This affects the maintenance of Government Furnished Equipment (GFE) used in facilities as part of the mission system. In addition, any proposed changes in configuration for such equipment must be properly coordinated.	SECNAVINST 5000.2A
NAVSEAINST 4130.16	Identification Practices for Systems, Equipment, Computer Software, and Firmware	16 Jun 92	This instruction prescribes policy and procedures for obtaining unique system and equipment identifiers for purposes of: (a) configuration identification, and (b) cataloging, storing, and issuing of material.	
NAVSEAINST 3960.2D	Test and Evaluation	22 Apr 88	This instruction prescribes the policies and procedures for acquisition programs of the Naval Sea Systems Command (NAVSEA).	OPNAVINST 3960.10C

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
NAVSEAINST 3560.1	Delivery of Naval Embedded Computer Programs		This instruction establishes policy and procedures for delivering software to ships and other afloat units.	
SPAWARINST 5200.22	Naval Electronic Systems Command Computer Resources Acquisition Management	14 May 90	This instruction established basic policy of the Naval Electronic Systems Command with regard to computer resources acquisition management, placing particular emphasis on the development of embedded computer software products. Its application is appropriate for systems using DOD-STD-2167 and 2167A.	
SPAWARINST 5200.23 (Ch-1)	SPAWAR Computer Software Life-Cycle Management Guide	1 Mar 79	The purpose of this guide is to provide information to the Space and Naval Warfare Systems Command (SPAWAR) Program Manager to allow him or her to understand and be able to meet sound software development practices for a SPAWAR software system acquisition. Its application is appropriate for systems using MIL-STD-1679.	MIL-STD-1679

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
SPAWARINST 4200.26	Procedures for Effective Acquisition of SPAWAR Systems, Equipment, and Support Services	22 Apr 88	this instruction establishes policies and assigns responsibilities to promote effective material acquisition through the use of non-development items, NBIs to fulfill Navy requirements for SPAWAR systems.	SECNAVINST 5000.2A
SPAWARINST 3960.3B	Test and Evaluation	31 Oct 89		
SPAWARINST 5100.5C	SPAWAR Systems Safety Program	29 Nov 87	This instruction establishes policy and assigns responsibilities for a formal system safety process within NAVAIR. The instruction applies to programs for which NAVAIR provides system safety engineering support for systems, subsystems, equipments, and facilities. The system safety program directs attention to early recognition of hazards and their elimination or control for SPAWAR Systems.	

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
SPAWARINST 4000.6D	Integrated Logistics Support Policy and Responsibilities	21 Jul 83	This instruction establishes policy and procedures for sitting up an ILS program within SPAWAR. This includes establishing a computer resources support activity program.	
SPAWARINST 4000.13A	ILS Assessment and Certification Procedures for System Acquisitions	31 May 88	This instruction establishes policy and procedures for conducting SPAWAR LAR _____ computer resources support is one of the elements _____.	
SPAWARINST 5231.1	Non tasked Information Systems, responsibilities, Policies, and Procedurals	1 Apr 87	This instruction establishes policy and procedures for acquiring non-MCCR systems in SPAWAR.	

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
SPAWARINST 4130.3A	SPAWAR Configuration Management Policy and Procedures	22 Apr 88	This instruction establishes policy and procedures for developing and conducting a configuration management program for SPAWAR systems. Information on GWCM requirements is also included.	
SPAWARINST 4000.6D	Integrated Logistics Support Policy and Responsibilities	21 Jul 83	This instruction establishes policy and procedures for establishing an Integrated Logistics Support (ILS) program within SPAWAR. This activity includes establishing a computer resources support program.	
SPAWARINST 4000.13A	ILS Assessment and Certification Procedures for System Acquisitions	31 May 88	This instruction establishes policy and procedures for conducting SPAWAR LAR audits. Computer resources support is one of the elements audited.	

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTIONS/COMMENTS
AV 2000A	Format for Naval Air Systems Command Avionic Equipment Performance Specifications	14 Nov 83	This document provides the format of the performance specification for equipment. Each paragraph explains its use and available options. The format includes information on the marking of software documentation, defines software media (tape, disks), and defines applicable interface data that may include bus structures and formats. In addition, material is provided relative to the programming of microprocessors and for the use of (HOC) in the design.	
AV 10000A, Supplement 1	Examples of Paragraphs for Specifying Avionics System Performance	2 Jun 82	This document supplements the AV-1000 format document by providing specific examples of paragraph wording, tables, and figures. It contains several examples of not only general information on computer software, but specifics on support programs and operational programs. The functional description of these capabilities is in performance terms given.	
AV 10000B	Format for Naval Air Systems Command Avionic System Performance Specifications for Weapons Systems	2 Jun 83	This document contains the basic specification format for an overall weapon system specification, based upon MIL-STD-961 and MIL-STD-490, together with examples of the use of the format for specific systems. It provides extensive information on documentation for Avionic Systems (computer) software and an outline of a computer program abstract.	MIL-STD-961 MIL-STD-490

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
AVION Instruction 5235.1	Avionics Software Metrics	18 Jun 92	This instruction defines a basic set of software metrics to be collected in support of weapon system procurements for the Naval Air Systems Command.	DODI 5000.2
MIL-STD-499A (USAF)	Engineering Management	1 May 74	This standard has been developed to assist Government and contractor personnel in defining the systems engineering effort in support of defense acquisition programs. This standard applies to internal DOD systems engineering as well as joint Government-industry applications for Government contracts. The fundamental concept of this standard is to present a single set of criteria against which all personnel may propose their individual internal procedures as a means of satisfying engineering requirements. A draft version of MIL-STD-499B is in the making.	DOD-STD-2167A DODI 5000.2
MIL-STD-881A B	Work Breakdown Structure	25 Mar 93	This standard contains Work Breakdown Structures (WBSs). These are the defined elements of work that structure a complete weapons system and its logistics into a numeric outline by which budgets can be planned and costs can be accumulated into meaningful categories. It provides for the development of WBSs for software development.	DODI 5000.2

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
MIL-STD-882B	System Safety Program Requirements	1 Jul 87	This standard contains seven tasks (300 Series Tasks) regarding the analysis of software safety hazards. These tasks roughly correspond to the various phases of DOD-STD-2167A. MIL-STD-882C, recently approved, does not contain these tasks.	
MIL-STD-498 Draft	Military Standard Defense Software Development and Documentation		This standard will provide DOD and its industry partners with a single contractual framework for developing DOD software assets. The standard will advance DOD "state of the practice" beyond our current standards (DOD-STD-2167A and DOD-STD-7935) by addressing (CASE), handling Computer-Aided Software Engineering various life-cycle development models, and providing a framework for metrics, reuse, and security. The standard will also provide a unified DOD voice in articulating software requirements for related commercial standards efforts that are recently underway.	
MIL-STD-490A	Specification Practices	4 Jun 85	This standard sets forth practices for the preparation, interpretation, change, and revision of program-unique specifications prepared by or for DOD. A draft version of MIL-STD 490B is in the making.	DODI 5000.2

NUMBER	TITLE	DATE	ABSTRACT	RELATED INSTRUCTION/COMMENTS
MIL-Q-9858A	Quality Program Requirements	16 Dec 63	This specification requires the establishment of a quality program by a contractor to assure compliance with the requirements of a contract. It provides a framework for government and contractor roles.	DOD-STD-2168 DODI 5000.2
ANSI/MIL-STD-1815A	Ada Programming Language	22 Jan 83	This is the Ada language specification for Ada 83. Currently, the Ada 83 standard is under revision. The draft Ada 9X standard ready for the first ballot will be released by American National Standards Institute (ANSI) and International Organization for Standardization (ISO) in the last quarter of 1993. ISO approval of the revision is expected in 1994. This is the language specification for Ada 83. Currently Ada 9X revision underway to update the Ada 83 standard. ISO and ANSI approval is anticipated for 1994.	DODD 3405.1 DODI 5000.2
MIL-STD-973	Configuration Management	17 Apr 92	This standard defines system-level configuration management requirements, as required. It includes software practices for use throughout the software development life cycle and post-deployment support. This standard is basically for the Government's internal use; however, the Government contractor may be required to conduct in accordance with MIL-STD-973.	DODI 5000.2

Appendix C

The Maturity Framework

The maturity framework for characterizing the status of a software process identifies five maturity levels. This framework is intended for use in conjunction with an assessment methodology and a management system. Assessment provides a way to identify the organization's specific maturity status, and the management system establishes a structure for actually implementing the main actions needed to improve the organization.

A maturity level is a well-defined evolutionary plateau on the path toward becoming a mature software organization. Each level is a layer in the foundation for continuous process improvement. The five maturity levels in the Software Engineering Institute (SEI) Capability Maturity Model (CMM) are defined as follows:

- **Initial**—At this level, the organization typically does not provide a stable environment for developing and maintaining software. The software process is constantly changed or modified as the work progresses. Until the process is under statistical control, orderly progress in process improvement is impossible.

- **Repeatable**—Basic project management processes are established to track commitments, cost, schedule, changes, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

- **Defined**—The software process for both management and engineering activities is documented, standardized, and integrated into an organization-wide software process. All projects use a documented and approved version of the organization's process for developing and maintaining software. At this point, it is probable that advanced technology can be usefully introduced.

- **Managed**—Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. At this level, the most significant quality improvements begin to appear.

- **Optimized**—With a measured process in place, the foundation exists for continuing improvement and optimization of the process. Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technology.

The subsections below contain additional information on the Initial and Repeatable Processes.

C.1 INITIAL PROCESS

The following paragraphs on the Initial Process are from the Carnegie-Mellon University/Software Engineering Institute (CMU/SEI) publication, *Characterizing the Software Process: A Maturity Framework*, by Watts Humphrey (CMU/SEI-87-TR-11, June 1987.)

> The Initial Process could properly be called ad hoc or chaotic. Here, the organization typically operates without formalized procedures, cost estimates, and project plans. Tools are not well integrated with the process or uniformly applied. Change control is lax and there is little senior management exposure or understanding of the problems and issues. Since problems are often deferred or even forgotten rather than solved, software installation and maintenance often present serious problems.
>
> While organizations at the Initial Process may have formal procedures in place for project control, there is no management mechanism to assure that they are used. The best test is to observe how such an organization behaves in a crisis. If it abandons established procedures and reverts to merely coding and testing, it is likely to be at the Initial Process. In essence, if the process is appropriate, it must be used in a crisis and if it is not appropriate, it should not be used at all.
>
> One key reason why organizations behave in this chaotic fashion is that they have not gained sufficient experience to understand the consequences of such behavior. Since many effective software actions such as design and code reviews or test data analysis do not appear to directly support shipping the product, they seem expendable. It is much like driving an automobile. Few drivers with any experience will continue driving for very long when the engine warning light comes on, regardless of their rush. Similarly, most drivers starting on a new journey will, regardless of their hurry, pause to consult a map. They have learned the difference between speed and progress. In software, coding and testing seem like progress but they are often only wheel spinning. While they must be done, there is always the danger of going in the wrong direction. Without a sound plan and a thoughtful analysis of the problems, there is no way to know.
>
> Organizations at the Initial Process can advance to the Repeatable Process by instituting basic project controls. The most important are:

1. **Project Management.** The fundamental role of a project management system is to ensure effective control of commitments. This control requires adequate preparation, clear responsibility, a public declaration, and a dedication to performance. Software project management starts with an understanding of the magnitude of the job to be done. In all but the simplest projects, a plan must be developed that lays out the most attainable schedule and the resources required. In the absence of such an orderly plan, the commitment to a schedule will be no more than an educated guess.

2. **Management Oversight.** A suitable disciplined software development organization must have corporate oversight. This oversight includes reviewing and approving all plans for major developments before the organization commits to the development officially. Quarterly reviews for each project must be conducted to determine facility-wide process compliance and quality of performance in the field; track schedule cost trends and computing service; and check quality and productivity goals. The lack of such reviews typically results in uneven and generally inadequate implementation of the process as well as frequent overcommitments and cost surprises.

3. **Product Assurance.** A product assurance group is charged with assuring management that the software development work is being done as it should be. To be effective, the assurance group must report directly to senior management and must have sufficient resources to monitor performance of all key planning, implementation, and verification activities. The size of the product assurance group generally is between 5% and 10% of that of the development organization.

4. **Change Control.** Control of changes in software development is fundamental to maintaining business and financial control as well as to technical stability. To develop high-quality software on a predictable schedule, the requirements must be established and maintained with reasonable stability throughout the development cycle. Changes will have to be made, but they must be managed and introduced in an orderly way. Although occasional changes are essential, evidence indicates that most changes can be deferred and phased in at a subsequent point. If change is not controlled, orderly testing is impossible and no quality plan can be effective.

C.2 REPEATABLE PROCESS

Organizations at the Repeatable level can advance to the Defined level by

instituting additional process controls. The most important controls are as follows:

- Standard processes for developing and maintaining software across the organization, as well as software engineering and software management processes, must be documented and must be integrated into a coherent whole. The organization must start exploiting effective software engineering practices when standardizing those software processes. A well-defined software process will provide a good view of technical progress on all projects as a result of the integration of all engineering activities related to those projects.

- The organization must establish and make effective use of a Software Engineering Process Group (SEPG) to facilitate process definition and improvement efforts.

- The organization must establish an organization-wide training program to ensure that all practitioners and managers acquire the necessary knowledge and skills required to perform their tasks successfully.

- The organization must establish good control of product lines, cost, schedule, and functionality and methodically track software quality and improvement. These control measures will help all personnel to arrive at a common understanding of processes, roles, and responsibilities.

Appendix D

Cost Estimation Studies

In April 1989, the Illinois Institute of Technology Research Institute (IITRI) completed a study, sponsored by the Ada Joint Program Office (AJPO), for the U.S. Air Force Cost Center (AFCSTC) and the U.S. Army Cost and Economic Analysis Center (USACEAC). The study assessed the accuracy of the software cost models for Ada software cost estimation. The six cost estimation models reviewed were as follows:

- Ada-Specific Models:
  - Ada COCOMO (Initial Operating Capability)
  - SoftCost-Ada
- Non-Ada-Specific Models:
  - PRICE S
  - SYSTEM-3
  - SPQR/20
  - Software Architecture Sizing and Estimating Tool (SASET).

An essential part of the research was a test case study in which the costs models were applied to a database of eight completed Ada projects. The analysis compared the projections on schedule and level of effort from each model as well as nominal run results to the actual project resources expended by the software developer. Results for each model were evaluated for accuracy and consistency in each of the following categories:

- Overall effort
- Overall schedule
- Government contracts
- Commercial contracts
- Command and control applications
- Tools and environment applications.

Model results were also evaluated based on the approach to the project and the personnel's experience with Ada. Interestingly, no correlation was found between the performance of the models and language considerations of the models described in the study. The test case study results demonstrated the benefits of using cost models to help the estimator predict resource requirements for a new development, but they

did not validate the need for Ada-specific models. Although SoftCost-Ada was the most accurate model on all dimensions, non-Ada-specific models were comparable in terms of accuracy and consistency. Of greater interest, however, is that the results suggest users should consider the following factors to determine which models should be applied to estimate Ada software costs:

- The amount of information available about the project and the developing organization
- The customer
- The type of application.

Before the IITRI study, the MITRE Cost Analysis Technical Center (CATC) conducted research to study the "early returns" of developing software in Ada. The research showed that the productivity for first-time Ada developments was not significantly different from that for non-Ada developments, Constructive Cost Model (COCOMO) semidetached developments, which supported the early claims of programming language comparability. This research provided calibrated equations and recommended guidelines for estimating the costs and schedule for Ada projects that are considered to be organic and semidetached developments.

Subsequently, in 1991, a follow-on research effort was initiated to expand the CATC database of embedded Ada projects, to develop new models for estimating level of effort and duration of Ada developments, and to expand the CATC's knowledge of the issues and cost impacts of programming in Ada. This research produced improved parametric models and equations for estimating the level of effort for Ada software developments and developed improved methods for quantifying the errors in these models. The study also developed appropriate guidelines for estimating costs and schedules of embedded Ada developments.

The most significant finding of this study was that productivity was better on embedded software developments where Ada was used as the programming language than on developments where non-Ada languages, such as FORTRAN, JOVIAL, or C++, were used. The calibrated level-of-effort models for Ada and non-Ada show significant differences in their respective predictions of level of effort for large projects. Although this result is based on a limited data set, the findings support claims that, for large projects, Ada software is less costly to develop than non-Ada software.

For embedded Ada developments, the following coefficients should be used when using the COCOMO equations for estimating software development cost and duration:

SM = 8.2 KEDSI

CM = 4.8 SM0.29

where,

SM = development effort in staff-months

CM = project duration in calendar months

KEDSI = software size in thousand of delivered source instructions (KEDSI ranges from 16 to 415)

The linear form of the level-of-effort model for embedded Ada developments represents a significant difference from traditional level-of-effort models for non-Ada software developments. For large Ada projects, the non-Ada level-of-effort models significantly overestimate the development effort. Although additional data on large Ada projects are needed, these findings strongly indicate that developing large programs may be less costly in Ada than in other High Order Languages (HOLs).

The MITRE research also resulted in the development of new schedule equations that incorporate software size as the independent variable. For embedded Ada developments, the following model can be used as an alternative method for estimating software development duration:

CM = 11.8 KEDSI0.23

It should be noted that the above equations were developed using a limited set of projects ranging in size from 16 to 415 KEDSI. The application of these models to projects outside this range should be avoided.

For semidetached Ada developments, the following equations should be used for estimating software developments costs and duration:

SM = 7.4 KEDSI (KEDSI ranges from 2 to 72)
CM = 6.0 SM0.22
CM = 10.3 KEDSI0.20

Again, the calibrated equations for semidetached developments were based on projects spanning a software size range from 2 to 72 KEDSI; therefore, analysts should exercise caution if using these semidetached equations for estimating outside of this range.

It is important to note that the software size inputs to the level-of-effort models and new models for estimating schedule are based on Ada statements (terminal semicolons) rather than physical lines of code. For sizing purposes, one Ada statement can be considered equivalent to one statement in other HOLs such as FORTRAN and JOVIAL. When analysts obtain sizing estimates for use in the estimating models, care should be taken to ensure that the inputs are in Ada statements or terminal semicolons rather than in lines of code. Where possible, analogies should be used when specific project information is available.

When little information is known about the project, or if a quick, rough estimate is needed, productivity can be used to estimate software development effort. For these cases, the database of Ada projects indicates a productivity range of approximately 75 to 160 Equivalent Delivered Source Instructions Per Staff-Month (EDSI/SM) for embedded Ada and approximately 110 to 230 EDSI/SM for semidetached Ada. When an average productivity rate is used to develop rough point estimates of effort, an average productivity of 122 EDSI/SM is appropriate for embedded projects, whereas an average productivity of 160 EDSI/SM is appropriate for semidetached projects. Again, it is strongly recommended that the database of Ada projects be used to develop an analogy when specific project information is available.

In addition to the software estimation models identified by the IITRI and MITRE studies, several other models are currently used and recommended for use by programs within the Department of the Navy. The following automated models are popular among the acquisition and development communities for forecasting program software costs because of their ease of use, reliability, and validity:

- SASET
- Revised Intermediate COCOMO (REVIC)
- Software Evaluation and Estimation of Resources (SEERÖ)
- COSTAR
- SEERÖ SEM (Software Estimation Model)
- SEERÖ SSM (Software Sizing Model).

SEERÖ is a registered trademark of SEER Technologies Division, Galorath Associates, Incorporated, Marina del Rey, California.

SASET is available upon request to programs for their use. Also, SASET is being modified to accommodate Function Point analysis. The Navy Center for Cost Analysis (NCA) can provide programs with the points of contact in the other Services who have other software cost estimating models, such as REVIC. The National Aeronautics and Space Administration's (NASA's) version of AdaCOCOMO, COSTMODL, also is available; however, the U.S. Air Force point of contact does not give it high marks for Automated Information Systems (AISs). Also, NASA's AdaCOCOMO is tuned for tactical applications. NCA also does the Independent Cost Estimates for Navy AIS programs.

Appendix E

Example of Metric Wording for Use in a Contractual Document

Software Management Metrics Requirements

The contractor shall include graphs of Software Management Metrics (SMM) in the Software Developmental Status Report (SDSR). The *x*-axis of each graph shall contain the calendar months of the program and shall depict the times of System Requirements Review (SRR), System Design Review (SDR), Preliminary Design Review (PDR), and Critical Design Review (CDR). Should SMM data change as SDSRs are presented, the contractor shall always show the original estimate together with the current estimate and indicate the changes since the last estimate.

SMM data shall be depicted for all software, regardless of whether the prime contractor and/or subcontractors (if any) are involved in the development and whether the software is newly developed, existing, or reused.

The information shown shall be as specified below.

1. Software Size

The contractor shall use an automated tool to estimate the size of the software that needs to be developed and shall report this estimated size.

During actual software development, the contractor shall report the Source Lines of Code (SLOC) elements in accordance with the information provided in the attachment to this appendix (to be supplied by contracting organization). SLOC metrics shall be provided for each Configuration Item (CI) for each programming language used. The contractor shall utilize an automated Code Counting Program (CCP) to provide the SLOC metrics results.

On a single graph, the contractor shall show the current values of total, new, reused, and modified SLOC counts.

2. Design Complexity

As software is developed, for each programming language used, the contractor shall use the appropriate static analyzer of the Verilog Logiscope tool to show flow graphs, call graphs, and Kiviat diagrams for each CI.

3. Software Personnel

The contractor shall record the number of engineering and management personnel supporting software development in experience categories of 1 through 3 years, 4 through 8 years, and 9 or more years. Software system planning, requirements definition, design, coding, testing, documentation, configuration management, and Quality Assurance (QA) personnel shall be included.

Each contractor development organization must provide graphs showing planned and actual personnel in the various experience categories.

4. Software Volatility

The contractor shall provide three different graphs showing software volatility on the *y*-axis. One graph shall contain the total number of "shall" statements (requirements) in the Software Requirements Specifications (SRSs) and the cumulative number of requirements changes (including additions, deletions, and modifications). A second graph shall contain new and cumulative Software Requirements Changes (SRCs), which are the number of unresolved requirement and/or design issues. A third graph shall depict Software Action Items (SAIs) that have been open from 1 to 45 days, 46 to 90 days, or over 90 days.

5. Computer Software Unit Development Progress

The contractor shall graph the progress made in Computer Software Unit (CSU) development against initial plans. This progress shall be reported to show monthly planned versus actual progress of the number of CSUs designed, tested, and integrated.

6. Testing Progress

The contractor shall record and graph the progress of CI and system testing against initial plans and the degree to which the software is meeting requirements. One graph shall depict the number of CI tests planned and passed, together with the number of system tests planned and passed. A second graph shall depict the number of new Software Problem Reports (SPRs) per month and the SPR density, which is the cumulative number of SPRs per 1,000 SLOC. A third graph shall depict the cumulative number of open SPRs and the number of SPRs that have been open from 1 through 45 days, 46 through 90 days, or over 90 days.

7. Build Release Metric

The contractor shall present a graph that contains each build, or release, of the software, showing the number of originally planned versus currently planned CSUs for each release.

8. Computer Resource Utilization

The contractor shall record the utilization of each target computer resource, including memory (all types), Input/Output (I/O) channels, I/O bandwidth, processor throughput under various extreme system loads, expected "normal" system load (including I/O), and memory use during processing times. Utilization metrics shall be proposed by the contractor and approved by the Government. The data shall show the planned versus actual utilization for each target computer resource. In addition, the contractor shall report on availability and use of host development station(s) to show planned versus actual usage.

Appendix F

Software Tool Descriptions

Editor

The editor is a tool for text manipulation. When computers were in their infancy, source program text was entered by paper tape or punched card. Today, editors are sophisticated interactive screen/window-management tools. Modern editors are used not only for creating or modifying source text but also for viewing or modifying files produced by other tools.

An editor is used primarily to create or modify source program text. The product of the editor is a file that contains the source program statements. Because these program files will always have to be compiled, the advantage was recognized of having a language-specific editor, a tool with some built-in specific language requirements. These editors simplify the entering of program text and sometimes perform on-line error checking.

In its most elementary form, a language-specific editor may have special options to assist in formatting the source text. An example for FORTRAN would be automatically starting a line in column 7 whenever the first character was alphabetic, thus preventing text from being placed in the field reserved for line numbers.

Another form of a language-specific editor is the "syntax-directed editor," which is tightly linked to the programming language. Most modern languages require opening and closing statements for structured programming constructs. A syntax-directed editor can provide templates for these constructs. For example, the following template could be rapidly placed on the screen after typing "if":

```
if <condition>
then
....;
....;
....;
else
....;
....;
endif;
```

In addition, the syntax-directed editor can check the structure of the source text for compliance with the rules of the language. Thus, the efficiency of the edit-compilation process is improved because many programming errors are eliminated before compilation.

Compiler
A compiler is a program that translates a High Order Language (HOL) source program into its relocatable code equivalent. The term "host" refers to the computer that translates the source program, and the term "target" refers to the computer that will execute the compiled code. The term "cross-compiler" refers to the case in which the target computer is different from the host computer. In many DON applications, a source program is cross-compiled on a host computer (generally a commercial machine) for a militarized target computer that is embedded in a system.

Compilers are usually multiple pass programs that may process the source program or some intermediate form several times before completion. The output of the earlier stages is referred to as intermediate code. In some host computer systems, the intermediate code is used by other tools.

Compilers for real-time applications must produce code to fit in limited storage space. In addition, the execution speed on the target computer must allow all of the required functions to be computed in the assigned time.

Assembler
An assembler is a program that translates an Assembly source program into relocatable code. Note that, usually, a one-to-one correspondence exists between an Assembly source statement and a machine instruction. Assemblers allow the programmer to use relative addressing and then specify a starting location rather than having to specify each address in absolute terms. Most assemblers also allow the use of labels and other defined values and locations.

Assembly has been used frequently in Mission-Critical Computer Resources (MCCR) applications because it allows the programmer to optimize storage space and execution time. However, Assembly programs are difficult to test and expensive to maintain. Today, the use of Assembly is generally restricted to routines with exceptionally high performance requirements and to hardware-diagnostic software.

Linker
Source program modules, whether in Assembly or a HOL, usually are translated separately into modules of relocatable code. Once translated, the modules must be

linked together before execution. A linker is a program that creates a load module from one or more independently translated modules by resolving the cross-references among the modules.

Relocating Loader

Relocatable code contains relative addresses of machine instructions and data. This defers the assignment of absolute addresses until the program is ready for execution and allows the flexibility of placing the program in any contiguous block of storage. The linker creates a load module that leaves all addresses in relative form although it has resolved the cross-references between modules. The relocating loader is a program that executes on the host computer and translates the relative addresses into the absolute addresses; its output is an execution module. A bootstrap loader executes on the target computer and copies the execution module into its storage.

Run-Time Environment

The Run-Time Environment (RTE) resides on the target machine and provides a variety of services for application programs. Typical RTE functions are dynamic storage management, exception processing, input and output, and task scheduling. Because this environment is used for all application programs, it should be small and fast to minimize the overhead. The RTE usually is modularized according to the particular services it provides and is automatically configured when the execution module is created. Thus, if an application program does not need a particular service, that module is automatically omitted from the RTE.

Simulator/Emulator

When code is produced for a target computer that is different from the host, the problem of how to test the code must be considered. Testing on the target computer is usually difficult because the computer may still be under development, it may be being integrated with other embedded subsystems, or the number of target machines may be insufficient to support all of the programmers. Moreover, most embedded target computers have poor tools to support testing.

One solution to this problem is to build a software simulator or emulator of the target computer that executes on the host computer. A software emulator accepts the same data, executes the same instructions, and achieves the same results as the target machine. A simulator imitates selected features of the target computer but is not required to achieve identical results. The best tool is a target computer emulator that can operate in either batch or interactive mode. The execution speed of an emulator may be significantly slower than that of the target computer, but the emulator has many advantages. For example, the emulator can be time shared and used by everyone on the host computer. In addition, because the emulator is on the host computer, it is easy to generate test data, load the module and test data into the

emulator, and monitor the test while in progress. Long tests can be run in batch mode during off-peak hours.

In-Circuit Emulator

An in-circuit emulator provides the user with a means of executing a software program located in external Random Access Memory (RAM) rather than internal Read Only Memory (ROM) or Erasable Programmable Read Only Memory (EPROM). This allows easy and rapid modification of the programs being debugged during the testing cycle. When connected to the prototype system through the microprocessor socket, an in-circuit emulator can emulate, test, and trace the prototype system operation. The internal state of the microprocessor, including RAM, accumulator, internal working registers, and stack and status registers, can be observed and modified. Some in-circuit emulators allow the recording of data bus operations. This feature allows the engineer to capture N events before or after a failure or predefined occurrence.

Symbolic Debugger

A symbolic debugger allows a programmer to test a module by controlling the program execution on a target computer emulator or the target computer itself. With the symbolic debugger, the programmer can address the variables by using their source program symbols or names. The facilities usually provided include stopping execution at selected locations, moving by single steps in increments of source statements, watching the value of specified variables, tracing execution, examining the contents of variables, evaluating expressions, displaying the current sequence of routine calls, displaying the source corresponding to any part of the program, executing debug command procedures at break points, and calling procedures that are program parts.

Pretty Printer

A pretty printer is a program that automatically applies standard rules for formatting program source code. It will accept an input file and format the text to match a style guide. For example, a pretty printer for a block-structured language will produce a listing in which the indentation level of each block shows its nesting level. A pretty printer helps the programmer read and comprehend the program. After extensive program modifications, for example, it helps eliminate confusion about the program structure and nesting levels.

Host-to-Target Exporter

If the target machine is different from the host machine, it is necessary to have a tool to transmit the execution module from the host to the target. Standard communications software and hardware may be used, but these are rarely available for embedded machines.

It is desirable to have a flexible, high-bandwidth communications link between the host and target. If the link has a "pass-through" capability, then an interactive user of the host computer can run tests on the embedded computer from the same terminal. High bandwidth is desirable because large volumes of data must be exchanged between the host and target; for example, diagnostic software is typically sent to the target to test the target hardware, and test data and test results are also exchanged. A high-bandwidth communications link will reduce the time it takes to do these tasks and allow more time for testing.

Computer-Aided Software Engineering

Computer-Aided Software Engineering (CASE) refers to software tools that help automate parts of the software process across the life cycle. These tools are considered a part of a Project Support Environment (PSE). CASE tools support the activities associated with a specific part of the system software life cycle, such as requirements specification, coding, and testing. CASE tools also can support project management activities across the life cycle. Recently, many tools have emerged that support software requirements specification and high-level design. CASE tools can help users organize, document, and generate a specification. Some of the more advanced tools also can execute simulations of the specification and, to a certain degree, generate Ada code or code fragments that fulfill the developed requirements and design. Although many CASE tools will examine the specification and high-level design for consistency and completeness, current CASE tools have widely varying degrees of functionality and maturity. The future of CASE tools is bright and the potential benefits great. Many tools are immature, however, and contractors' claims regarding the capabilities of their tools are often exaggerated. Some CASE tools have difficulty scaling-up to support large software developments (e.g., more than 100,000 Source Lines of Code [SLOC]). Finally, few CASE tools are compatible with each other with regard to method of data transfer or integrated execution.

Many CASE issues are similar to issues surrounding the introduction of Ada. CASE will help impose discipline on the software process, provide better visibility into the software, and encourage the use of modern methods and practices. Three primary benefits of using CASE on a project are improved product documentation, improved project communication, and enforcement of a consistent design and requirements methodology.

Many of the issues surrounding the adoption and use of CASE are organizational, not technical, issues. The organization must have a well-defined software engineering methodology in place before the benefits of CASE tools can be realized. Use of CASE tools often requires a pervasive change in an organization. First, an absolute

and strong management support of and commitment to CASE use are needed. Second, selection of high-quality personnel and extensive training are necessary. Third, the resistance to change must be overcome.

In some cases, the initial adoption of CASE requires a large capital investment and, most likely, schedule expansion. This up-front cost in terms of dollars and schedule will be recovered when the lower maintenance costs are realized.

CASE tools are emerging on the market that maintain the specification at a high level and automatically generate Ada code. The intent is to make all changes at the specification level and not at the code level. This technology promises to provide many benefits to software engineering. CASE will surely have an increasing impact on programs developed in Ada and the Ada software development environment.

Appendix G

Application Portability Profile (APP) Services

This appendix identifies current and emerging standards associated with the service areas addressed in the National Institute of Science and Technology (NIST) Application Portablility Profile (APP) and DOD Technical Architecture Framework for Information Management (TAFIM).

G.1 OPERATING SYSTEM SERVICES

The following subsections discuss some recommended operating system services.

G.1.1 Kernel Operations API

FIPS PUB 151-2 Portable Operating System Interface (POSIX)—System Application Programming Interface (API). Kernel operations provide low-level services necessary to create and manage processes, execute programs, define and communicate signals, define and process system clock operations, manage files and directories, and control Input/Output (I/O) processing to and from external devices.

G.1.2 Operating System Commands and Utilities API

Planned FIPS PUB on POSIX—Part 2. Commands and utilities include mechanisms for operations at the operator level, such as comparing, printing, and displaying file contents; editing files; pattern searching; evaluating expressions; logging messages; moving files between directories; sorting data; executing command scripts; scheduling signal execution processes; and accessing environment information.

G.1.3 Operating System Real-time Services API

Amendment 1: Realtime Extension P1003.4 Draft 12. This document provides the operating system extensions needed to allow incorporation of real-time application domains into the Open Systems Environment (OSE). The extensions define the application's interface to basic system services for I/O, file system access, and process management.

G.1.4 Operating System Security API

Security Interface for POSIX (IEEE P1003.6 Draft 11). Security considerations are specified in terms of data encryption mechanism, access control, reliability control, system logging, fault tolerance, and audit facilities.

G.2 HUMAN-COMPUTER INTERFACE SERVICES

The following subsections discuss the recommended Human-Computer Interface (HCI) services.

G.2.1 Graphical User Interface API

Proposed FIPS PUB User Interface Component of the APP (MIT XWindow System). The MIT XWindow System is the Federal standard for Graphical User Interfaces (GUIs) in the OSE. Its software has proven to be highly portable between various hardware platforms and operating systems. Because of its client-server architecture, the X client application can run on one system while the X server can be running on another system on a network.

G.2.2 Graphical User Interface Toolkit API

Draft Standard for Information Technology—XWindow System Graphical User Interface—Part 1: Modular Toolkit Environment (IEEE P1295.1). This specification supports writing portable applications with GUIs based on the XWindow System. It defines a source-code-level interface to an XWindow System toolkit GUI environment based on the OSF Motif Application Environment Specification User Environment volume.

G.3 SOFTWARE ENGINEERING SERVICES

The following subsections discuss certain recommended software engineering services.

G.3.1 Programming Language Ada

FIPS PUB 119 Ada. Ada is a general-purpose, high-level programming language. In addition, it provides strong data-typing, concurrence, and significant code-structuring capabilities. It is particularly suited to embedded real-time systems, distributed systems, highly reliable software development, and reuse of proven code.

G.3.2 Integrated Software Engineering Environment

Portable Common Tools Environment (PCTE): Abstract Specification, Standard ECMA-149, European Computer Manufacturing Association (ECMA). Integrated Software Engineering Environments (ISEEs) and tools include systems and programs that assist in the automated development and maintenance of software. These include, but are not limited to, tools for requirements specifications and analysis, for design work and analysis, for creating and testing program code, for documenting, for prototyping, and for group communication. The interfaces among these tools include services for storing and retrieving information about systems and exchanging this information among the various programs in the development environment. PCTE can provide for this data repository functionality.

G.3.3 Other Programming Languages

See the NIST APP for FIP-PUBs on Other Programming Languages.

G.4 DATA MANAGEMENT SERVICES

The following subsections discuss certain recommended data management services.

G.4.1 Relational Database Management System Interface

Planned FIPS PUB 127-2 Database Language Structured Query Language (SQL). FIPS SQL provides data management services for definition, query, update, administration, and security of structured data stored in a relational database. A relational database is appropriate for general-purpose data management, especially applications requiring flexibility in data structures and access paths; it is particularly desirable where there is a substantial need for ad hoc data manipulation for data restructuring. The security interface for granting and revoking privileges does not specify a secure DBMS; only its interface.

G.4.2 Data Dictionary or Directory System

FIPS PUB 156 Information Resource Dictionary System (IRDS). Data dictionary or directory services consist of utilities and systems necessary to catalog, document, manage, and use metadata (information about data).

G.4.3 Distributed Data Access

Remote Database Access (RDA) ISO/IEC 9579:1993. RDA is used to establish a remote connection between an RDA client, acting on behalf of an application program or a client data manager, and an RDA server, interfacing to a process that controls data transfers to and from a database. The goal is to promote the interconnection of applications and the interoperability of Database Management Systems (DBMSs) among heterogeneous environments.

G.5 DATA INTERCHANGE SERVICES

The following subsections discuss certain recommended data interchange services.

G.5.1 Data Interchange

Open Document Architecture (ODA)/Open Document Interchange Format (ODIF)/ Open Document Language (ODL) [ODA/ODIF/ODL] ISO 8613:1989. ODA is a framework that enables users to interchange the logical structure, content, presentation style, and layout structure of documents from one application to another. ODIF is an encoding scheme for documents suitable for interchange between applications. ODL is a generic Standard Generalized Markup Language (SGML) encoding for ODA documents to enter an SGML database or publishing environment.

G.5.2 Document Interchange

FIPS PUB 152 Standard Generalized Markup Language (SGML). SGML is a generalized grammar used to write data type definitions for describing document types and styles.

G.5.3 Page Description Language

Planned FIPS for Standard Page Description Language (SPDL) ISO/IEC DIS 10180. SPDL defines a language for representing images that are to be displayed on a screen, printed on an output device, or transmitted through communications media from one application to another.

G.5.4 Manuscript Markup Interchange

Electronic Manuscript Preparation and Markup (EMPM) American National Standards Institute/National Information Standards Organization (ANSI)/(NISO) Z39.59-1988. EMPM is a specialized document type definition that includes an architecture-encoded SGML suitable for the interchange of the logical structure of books, articles, and serials.

G.5.5 Graphics Data Interchange

Computer Graphics Metafile (CGM), FIPS PUB 128. Graphics data are specified in terms of a file format that can be created independently of device requirements and translated into the formats needed by specific output devices, graphic systems, and computer systems.

G.5.6 Graphic Product Data Interchange

FIPS PUB 177 Initial Graphics Exchange Specification (IGES). IGES standardizes the representation of specific types of complex graphic objects and attributes for data interchange. Information typically associated with computer-aided design and manufacturing (CAD/CAM) can be described.

G.5.7 Product Life Cycle Data Interchange

Standard for the Exchange of Product Model Data (STEP) Draft Proposed ISO 10303. STEP is an advanced form of representing complex data objects for interchange. It is used in total life cycle descriptions of engineered products that can be implemented on advanced manufacturing systems. This includes specifications of products throughout the stages of their lifetimes.

G.5.8 Electronic Data Interchange

FIPS PUB 161 Electronic Data Interchange (EDI). EDI is a procedure in which instances of documents to be interchanged between separate organizations are converted to strictly formatted sequences of data elements and transmitted as messages between computers.

G.5.9 Spatial Data Interchange

FIPS PUB 173 Spatial Data Transfer Standard (SDTS). This standard is mandatory in the acquisition and development of Government applications and programs involving the transfer of digital spatial data among heterogeneous computer systems.

G.6 GRAPHICS SERVICES

The followint subsections discuss selected recommended graphics services.

G.6.1 Two-Dimensional Graphics API

FIPS PUB 120-1 Graphical Kernel System (GKS). GKS fulfills the requirement for a language to program two-dimensional graphical objects that will be displayed or plotted on appropriate devices (raster graphics and vector graphics devices).

G.6.2 Interactive and Three-Dimensional Graphics API

FIPS PUB 153 Programmer's Hierarchical Interactive Graphics System (PHIGS). PHIGS fulfills the requirements for a language to program two-dimensional and three-dimensional graphical objects that will be displayed or plotted on appropriate devices in interactive, high-performance environments, and for managing hierarchical database structures containing graphics data.

G.7 NETWORK SERVICES

The following subsections discuss certain recommended network services.

G.7.1 Communication API for Protocol Independent Interfaces

Protocol Independent Interfaces (PIIs) IEEE P1003.12, Draft 2.0. PII defines the protocol-independent application interfaces to enable one process to communicate with another local process or a remote process over a network.

G.7.2 Communication API for OSI Services

Open Systems Interconnection (OSI) Association Control Service Element (ACSE)/Presentation Application Program Interfaces IEEE P1238. OSI ACSE provides an API between applications and the OSI and presentation services.

G.7.3 File Transfer API

OSI API for File Transfer, Access, and Management (FTAM) IEEE P1238.1. FTAM is designed for use as a standard application interface for file transfer, access, and management applications.

G.7.4 Communications Protocols for OSI

FIPS PUB 146-1 Government Open System Interconnection Profile (GOSIP) Version 2. GOSIP protocols provide interoperability among applications in a heterogeneous network. GOSIP is based on the Open Systems Interconnection (OSI) standards, the

worldwide consensus standards for multivendor data communications based on OSI protocols.

G.7.5 Communication API for Integrated Digital, Video, and Voice
Application Software Interface (ASI) Version 1 for accessing and administering Integrated Services Digital Network (ISDN) services. ASI focuses on the definition of a common application interface for accessing and administering ISDN services provided by hardware commonly referred to in the vendor community as Network Adaptors (NAs).

G.7.6 Communication API for Integrated Digital, Video, and Voice
NIST Planned FIPS on Integrated Services Digital Network (ISDN). The proposed FIPS PUB compiles the existing North American ISDN User's Forum (NIUF) agreements for ISDN as developed and approved in the NIUF as of November 1990.

G.7.7 Remote Procedure Call
OSF Distributed Computing Environment (DCE) Remote Procedure Call (RPC) Component. Distributed computing services include specifications for remote procedure calls and distributed real-time support in heterogeneous networks.

G.7.8 Transparent Network Access to Remote Files
Transparent File Access (TFA) IEEE P1003.8, Draft 7. TFA includes capabilities for managing files and transmitting data through heterogeneous networks in a manner that is transparent (i.e., does not require knowledge of file location or of certain access requirements) to the user.

G.7.9 Network Management
FIPS PUB 179 Government Network Management Profile (GNMP), Version 1.0. The GNMP is the standard reference for all Federal Government agencies to use when acquiring Network Management (NM) functions and services for computer and communications systems and networks.

G.7.10 Electronic Messaging API
X.400-Based Electronic Messaging Application Program Interface (API) IEEE P1224.1, Draft 3. X.400 provides electronic mail interoperability among heterogeneous computer systems. X.400 is an international standard protocol definition. This API defines an interface between the user of a mail system and the mail system.

G.7.11 Directory Services API

Directory Services Application Program Interface (API) X.500 IEEE P1224.2. X.500 provides the interoperability of dictionary services among heterogeneous computer systems. The Directory Services (DS) API defines a standard directory service user agent interface to support application portability at the source-code level.

G.8 SECURITY SERVICES

No specifications are defined to support security services.

G.9 MANAGEMENT SERVICES

No specifications are defined to support management services.

G.10 NIST APP SPECIFICATIONS EVALUATIONS

The NIST APP evaluates recommended specifications for each of the APP services and summarizes some of the pros and cons of selecting each specification. The information is provided to managers, technical project leaders, and users to assist them in evaluating these specifications for inclusion in application or organizational profiles. Each specification is evaluated according to how well it meets the requirements of a specific criterion. The criteria are Level Of Consensus (LOC), Product AVailability (PAV), CoMPleteness (CMP), MATurity (MAT), STaBility (STB), De Facto Usage (DFU), and PRoblems/Limitations (PRL). Definitions of these criteria are provided in the NIST APP. Table G-1 presents the evaluations for selected specifications.

Table G-1. Evaluations of NIST APP Specifications

	LOC	PAV	CMP	MAT	STB	DEU	PBL
Operating System Services							
FIPS PUB 151-2 POSIX	+	+	+	+	+	+	+
POSIX SHELL IEEE 1003.2-1992	0	+	+	+	+	+	+
Real-Time IEEE P1003.4	0	-	0	+	0	-	-
Security IEEE P1003.6	-	-	-	0	-	-	-
Human-Computer Interface Services							
Proposed FIPS PUB 158-1 XWindow System	+	0	0	+	+	+	-
Draft XWindow Toolkit IEEE P1295.1	+	+	0	0	0	+	0
Software Engineering Services							
FIPS PUB 119 Ada	+	+	+	+	+	+	+
FIPS PUB 160 C	+	+	+	+	+	+	+
FIPS PUB 21-3 COBOL	+	+	+	+	+	+	+
FIPS PUB 69-1 FORTRAN	+	+	+	+	+	+	+
FIPS PUB 109 Pascal	+	+	-	+	+	-	-
ECMA PCTE	+	-	-	0	0	-	-
Data Management Services							
Planned FIPS PUB 127-1 SQL	+	+	+	+	+	+	+
FIPS PUB 156 IRDS	0	-	+	+	0	-	-
RDA	0	-	0	-	0	-	0
Data Interchange Services							
ODA/ODIF/ODL ISO 8613	+	-	+	0	0	-	0
FIPS PUB 152 SGML	+	-	0	+	+	-	0
SPDL ISO 10180	+	-	+	+	+	-	0
EMPM ANSI/NISO Z39.59	+	-	+	+	+	-	0
FIPS PUB 161 EDI	+	+	+	+	0	+	0
FIPS PUB 128 CGM	+	+	+	+	+	+	+
FIPS PUB 177 IGES	+	+	-	0	+	+	+
STEP ISP 10303	-	-	+	+	-	-	-
FIPS PUB 173 SDTS	0	0	+	+	+	+	+
Graphics Services							
FIPS PUB 120-1 GKS	+	+	+	+	+	+	+
FIPS PUB 153 PHIGS	+	+	+	0	+	+	+
Network Services							
PII API IEEE 01003.12	-	0	-	0	-	+	-
ASCE IEEE P1238	-	-	-	-	-	-	-
FTAM IEEE P1238.1	-	0	-	-	0	-	-
FIPS PUB 146-1 GOSIP 2	+	+	+	+	+	0	0
ISDN ASI	+	-	0	+	0	0	0
ISDN	0	-	0	0	0	0	+
DCE RPC	-	-	-	-	0	-	-
TFA IEEE P1003.8	-	0	+	+	/	-	-
FIPS PUB 179 GNMP	+	-	0	+	+	-	0
X.400 API IEEE P1224.1	0	-	+	0	+	0	+
X.500 API IEEE P1224.2	0	-	+	+	+	+	+

LEGEND + High evaluation 0 Average evaluation - Low evaluation

Appendix H
Ada Binding Products

VENDOR NAME	ASIS	GPEF	GPPF	GKS	M_WIN	PHIGS	POSIX	SQL	TCP/IP	X WIN
Advanced Technology Center				•		•				•
AETECH					•	•	•	•		•
Alsys	•	•					•	•		•
Competence Center Informatik, GmbH								•		
Convex Computer Corporation		•	•							
Digital Equipment Corporation				•	•			•		•
Encore		•	•							
Gallium Software, Inc.				•						
IBM		•						•		•
Ingres Corporation								•		
Integrated Computer Solutions								•		
Intermetrics								•		
MassTech		•								
Objective Interface Systems, Inc.										•
Oracle Corporation								•		
R.R. Software, Inc.					•					
Rational	•									•
SL Corporation		•	•				•	•	•	•
Silicon Graphics						•	•			
Software Technology, Inc.				•						
Sunrise Software International										•
Systems Engineering Research Corp.										•
Verdix Corporation										•

Key: • Indicates Ada binding product is available

Appendix I

Lessons Learned

Historically, the Department of Defense (DOD) has failed to take advantage of past experiences in developing software systems. The process of reviewing past experiences and formulating new decisions based on these experiences should be ongoing. It must start from the beginning of the process and continue through development and into deployment. The software process also should be adjusted for overall system size, technical complexity, and development phase.

This appendix describes decisions and evaluations made about software system developments that have led to lessons learned by the users of Ada in the commercial and Government sectors. Figure I-1 presents a matrix of the lessons learned by project in the following areas:

- Standards and policy
- Project management
- Development process
- Corporate knowledge and software development
- Training
- Resources and facilities
- Support environment and tools
- Reuse
- Project costs.

In some cases, the information presented in the project descriptions came from internally developed evaluations; in others, the information is based on externally developed assessments. Note that, except for editorial changes, the descriptions are reproduced here as submitted by the originators.

As the examples show, different projects often experienced similar problems and applied similar remedies. Frequently, a series of non-software-related errors culminated in the production of the wrong set of software components. The lessons learned from these and other project experiences may help managers of new projects to avoid such problems.

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
I.1	STRATCOM								
1	Visualize Ada as supporting an engineering approach to software			X			X		
2	Do not anticipate that use of Ada will decrease development time, particularly on the first few projects		X	X	X	X			X
3	Try to limit the number of variables in the development process on the first project		X	X		X	X		
4	Do not blame all problems on the software or Ada		X	X	X				
5	Select a small doable project for the first Ada effort		X	X	X	X			
6	Use tools		X	X	X	X	X	X	
7	Ensure that management understands that tools support software engineering, not programming	X	X	X	X		X	X	
8	Ensure that management focuses on the engineering aspects of Ada, not on Ada as a programming language		X		X	X			

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development	Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
9	Use a well-defined, repeatable process and methods for developing and maintaining software	X	X	X	X	X	X	X	X	X
10	Implement an active training program in both Ada and software engineering		X			X				X
11	Ensure that adequate resources are available		X		X		X	X		
12	Have adequate configuration management		X	X				X		
13	Assess software engineering capabilities before the project gets under way		X		X		X			
14	Do not expect to transform maintenance personnel into developers overnight				X	X				
15	Do not field products before they are ready		X		X					X
16	Tailor the Data Item Descriptions from MIL-STD-2167A	X	X							
17	Do not neglect documentation	X	X	X				X		

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
18	Use commercial-off-the-shelf products as well as reuse repositories, where available		X	X				X	X	
19	Beware of the software part received from outside repositories						X		X	
20	Do not make spot fixes without proper change control		X	X	X					
21	Have a test strategy that includes regression testing		X	X						
22	Have a thorough integration plan		X	X	X					
23	Ensure the Project Manager has authority to direct efforts and resources	X	X				X			
24	Clearly define the role of the Program Manager	X	X							
25	Ensure that the Project Manager of an Ada project is literate in software engineering and Ada				X	X				
26	Establish a system architecture and stick with it	X	X	X						

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development	Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
27	Apply configuration management and Quality Assurance (QA) to all parts of the system regardless of the language	X	X	X						
28	Do not flood the team with untrained people and expect them to get trained "free" on the job		X			X				
29	Ensure that upper management and CM personnel understand that they do not need to know how to program in Ada to manage projects written in Ada		X		X	X				
30	Provide periodic updates to non-Ada groups outside the project				X					
L2	WELLS FARGO									
1	To use Ada effectively, management at the highest level must be committed to Ada		X		X	X				
2	Ada requires an up-front investment, patience to wait for the up-front investment, and patience to wait for the pay back		X							X
3	Some criticism, pessism, and panic are almost inevitable		X		X					
4	Ada must be sold, and existing environments (and other languages) must be "un-sold"		X		X					

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development	Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
5	Support and commitment from other developers and users should grow over time as the language has time to prove itself				X					
6	Tools and adequate hardware to run the tools are needed						X	X		
7	Training is a must					X				
8	A software repository foundation is vital with both externally developed and internally developed software			X				X	X	
9	Ada can and should be used with other languages			X						
I.3	**B-2 AIRCREW TRAINING DEVICES**									
1	In the early Design Phase, capacity and performance estimates should target resources that provide at least a factor of 2 reserve	X	X	X	X		X			
2	When COTS language development products are to be used, project staff should expend as much effort as possible to determine which products best meet project needs	X	X	X	X			X		X
3	If well-thought-out engineering and management plans are implemented with enough resources, the project will be successful	X	X	X	X	X	X	X	X	X

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
I.4	**BOEING MILITARY AIRCRAFT**								
1	Careful technical management of the development of code directives, guidelines, and formulations enables project staff to maximize the amount of fully portable code that is produced	X		X		X		X	
2	The availability of a large suite of formal validation tests and use of an independent testing organization help ensure the basic qualifications of vendor-supplied products			X		X			
3	Lessons learned with respect to portability also apply to separate compilation			X			X	X	
4	Controlled use of Ada language constructs results in uniform and minimally complex code, thus maximizing readability								X
I.5	**COULTER ELECTRONICS**								
1	Look at the language and the constructs to be used and decide on an environment			X			X		
2	Evaluate your needs and then evaluate the compilers that run on your particular platform			X	X				
3	Look at external software programs that have to work with your particular program		X	X					

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
4	Ensure that the compiler has a method for accessing external hardware interfaces if the project equipment has such interfaces		X	X		X			
5	Remember that "optimization" should minimize the code size not just remove "dead code"		X	X					
6	Recognize that reuse can be a major factor in code development if looked at from the beginning		X	X				X	
L6	AN/UYS-2A PROJECT								
1	When selecting a host computer at the Application Development Facility, ensure the selected host computer supports Ada so that application developers do not need to purchase multiple hosts to develop application software		X		X	X	X		
2	When selecting Ada products, ensure that the Ada vendor can provide a full spectrum of products. Avoid using multiple vendors when possible		X		X	X	X		
3	Select a well-established Ada vendor who demonstrates willingness to help software developers move code to new versions of their compilers		X		X				
4	Establish a close working relationship with the Ada vendor and define project needs as early as possible. Plan Ada upgrades in a systematic and controlled manner		X		X				

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
I.7	**NAVAL RESEARCH AND DEVELOPMENT CENTER**								
1	The transition to object-based design and use of Ada enhanced productivity. It decreased integration time because there were fewer errors and less need to rework code		X	X	X	X	X		X
2	The use of automated tools and Ada have enhanced our ability to maintain developed products and their documentation		X	X	X	X	X		X
3	Project management should expect that at least 50% of the development time will be spent in the Requirements Analysis and Design Phases		X	X					
4	Attitude is a key factor in transitioning engineering personnel to modern software engineering and Ada			X		X		X	
I.8	**TACTICAL AIRCRAFT MISSION PLANNING SYSTEM**								
1	Before selecting vendor products, it is important to test them extensively to ensure they meet a project's specific needs		X	X	X		X		
2	Many details in the implementation process are not controlled by MIL-STD-1815A or the associated validation suite for the Ada language	X	X		X		X		
3	Staff should do up-front technical evaluations		X		X		X	X	X

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
I.9	**ADVANCED FIELD ARTILLERY TACTICAL DATA SYSTEM**								
1	Anticipate trouble with the Ada development tools/environment, no matter who is supplying them or when you get them						X		
2	Budget for training		X			X			X
3	Anticipate that original estimates for support hardware and facilities will have to be revised. In this project, original estimates quadrupled for support hardware and facilities					X			X
4	To accomplish the project successfully, ensure that both the contractor and Government teams are knowledgeable about and understand the rationale for all software-related topics		X	X	X				
5	Have the team develop a viable technical/ management plan and adhere to it so that requirements and design can be implemented correctly		X	X	X				
6	Report major problems up the line as encountered		X	X	X				
7	Do not mistakenly blame software development for failure. Careful scrutiny of many projects frequently shows that things other than software development are responsible for failure		X						

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development	Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
L10	ANDESY-2									
1	When external schedule constraints exist, the level of planning and execution analysis becomes much more critical		X							
2	Most of the "lessons learned" are related to the contract requirements		X	X	X					
3	Coordination frequently receives the least attention although it is one of the more important efforts		X	X						
4	For large projects, it is mandatory that an adequately sized, qualified Technical Directive Authority (TDA) Oversight Group be established and function for the duration of the project		X	X	X					
5	Software development planning and monitoring must be done from the onset of Full-Scale Development and should take a phased approach (i.e., "build a little, test a little").	X	X	X						
6	Even the best-made plans require changes during execution		X	X						
7	For large efforts that are geographically dispersed, the goal should be to strive for commonality of development environment, tools, procedures, and product structure		X	X	X		X	X		

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development	Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
LII	ADA LANGUAGE SYSTEM/NAVY									
1	For DON Standard Embedded Computer Resources (SECR) applications, top priority must be given to the real-time performance of the generated code		X	X	X		X	X		
2	Although actual software code production is only a relatively small portion of the total life-cycle, it is critical to have a reasonable level of performance within the tool set			X			X	X		
3	Each development effort should be managed under the assumption that there will be a formal production delivery to DON and a separate DON-controlled post-deployment phase	X	X	X						
4	Requirements must be understood, and both formal and informal checks on the program to meet these goals must be conducted throughout development	X	X	X	X					
5	Because post-deployment support will be DON's responsibility, it is critical to build an adequate in-house team that is thoroughly familiar with the product before acceptance	X	X	X	X	X				
6	Lack of full program funding commitment and support will have a negative impact on development plans. Be prepared to either alter the course of and/or extend delivery schedules		X	X						X
7	Producing a high-quality software-based product that meets its specified requirements is a difficult task	X	X	X	X	X	X	X	X	X
8	No product is truly exercised and tested until it reaches the target user community		X	X						

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development	Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
L12	AVIONICS PROJECT									
1	Ensure that software production/test modeling includes adequate time for the Requirements/Design Phase before accepting externally generated completion dates		X	X	X					
2	Be sure that requirements are fully defined and are traceable to test mechanisms	X	X	X	X			X		
3	Do not plan to use equipment that is under development unless absolutely necessary. Apply a risk engineering approach to those items that must be used, place items on a critical path, and monitor them closely		X	X			X	X		
4	Always assume that everything could go wrong and perform full risk engineering and management		X	X	X					X
5	Use a hands-on management approach from both the prime and Government perspectives, delineating clear lines of authority and responsibility for contractual requirements, especially for large projects	X	X	X	X	X				
6	Specify in the contract requirements that capabilities must be established early, with adequate resources and authority. Closely monitor progress	X	X	X	X	X	X	X		
7	Do not disregard the critical elements of the MIL-STDs unless it is technically and managerially necessary to use alternative means. Develop a system-wide integration plan and follow it	X	X	X	X					
8	Ensure that the schedule and communicate about and the possibility of independent DON test time for interim products. Also ensure that the resources are available to support regression testing		X	X	X		X			X

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
9	Do not disregard critical MIL-STD interim products in the contract requirements, and adequately plan for and execute the Government's role to ensure quality and delivery	X	X	X			X		
10	Ensure that adequate development support facilities exist. Existence of these facilities should be conveniently specified and monitored during the PRR process. Contingency plans should be available when and if problems develop	X	X	X		X	X		
11	Do not let events external to the schedule influence the program. Develop input and output criteria for major milestones and adhere to them. It is very easy to build the wrong software		X	X	X				
12	Where possible, use real production hardware and/or commercial prototypes to decrease the amount and scope of simulation		X	X			X	X	
13	Do not approve systems until requirements are met because when system requirements are not met and "as-built" systems are approved, the contractor is no longer responsible for fixing them	X	X	X					
L13	PEO-SSAS, PMS-414, SEA LANCE								
1	Use a consistent methodology throughout the program Requirements, Design, and Coding Phases to facilitate tracing requirements to the code	X	X	X	X	X		X	
2	Use a common Program Design Language (PDL) across the project. On medium- to large-scale systems, the PDL will contain a wide variety of differing coding techniques and code fragments	X	X	X		X			
3	Include and enforce a requirement for a minimum ratio of 50-50 comments-to-code in the contract, software development plan, or coding guide	X	X	X					

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development	Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
4	Use an automated format utility or equivalent software tool to ensure uniform code appearance. This can be applied by either QA or configuration management	X	X	X	X			X		
5	Develop a style guideline for the Ada code and PDL before doing any design work	X	X	X						
6	Use software metrics from the beginning, and define basic terminology between Ada and the selected software development standard	X	X	X						
7	Examine any documentation requirements and licensing agreements between the Government and the contractors regarding the use of third-party software and the way it is to be tested and identified	X	X	X					X	X
8	Early in the development process, have the contractor provide a detailed list of tools that will be used in the development process and specify the format that will be used for transfer of source code, executable code, and software documentation to the Government.	X	X	X			X	X		
I.14	NAVY WORLD WIDE MILITARY COMMAND AND CONTROL SYSTEMS SITE-UNIQUE SOFTWARE PROJECT									
1	*It is always safer to build and test incrementally*	X	X	X	X					
2	Planning for designing in reuse yields long-term benefits		X	X					X	
3	For large software undertakings, use of automated tools is mandatory		X	X	X			X		

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development	Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
4	Until the design baseline has been approved and frozen, it is inadvisable to initiate full-blown coding	X	X	X	X					
5	If a risk engineering approach (i.e., awareness, identification, technical management of alternative solutions) is taken to development, then it is possible to undertake technologically challenging developments		X	X	X					
I.15	EVENT-DRIVEN LANGUAGE/COBOL-TO-ADA CONVERSION PROGRAM									
1	Training is essential for both technical and management personnel		X	X		X				
2	Programmers require 4 to 9 months of training before they become proficient		X	X		X				
3	Military transfers often result in a loss of investment in Ada training		X			X				X
4	Systems originally written in languages that produce Ada that must be converted to Ada should be redesigned, not translated		X	X	X					
5	Ada facilitates reuse		X	X	X				X	
6	Ada lends itself to efficient coding and high programmer productivity		X	X						

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development	Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
7	Development tools are essential		X	X		X		X		
8	Development and maintenance time can be significantly reduced by applying software engineering principles and capitalizing on reuse	X	X	X	X	X			X	
L16	SHIPBOARD GRIDLOCK SYSTEM WITH AUTO-CORRELATION									
1	Before committing to large projects, the methods and tools to be used should be examined. Quantitative evaluation of the expanded resources should lead to better estimates for the work contemplated	X	X	X			X	X		
2	The Ada code itself will have major architectural and design impact on a system; therefore, the two must be worked on simultaneously		X	X						
3	A project should always try to build a little and test a little, building and testing the harder things first (e.g., system services and communications)	X	X	X						
4	A project should always attempt to involve the production hardware as early in the program as feasible	X	X	X			X			
5	The team must be well trained in the use of supplied tools, and the tools must work as advertised		X	X	X	X	X	X		
6	Adherence to good engineering practices is necessary when designing the system and its hardware and software	X	X	X	X	X				

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
7	Until more technological progress is achieved, the potential for large-scale software component reuse is limited		X	X				X	
L17	**SUBMARINE COMBAT CONTROL SYSTEM MK2**								
1	Ada experience and training are needed		X	X		X			
2	Support software, practices, and products need constant attention	X	X	X	X		X		
3	The need to interface with other language programs may constrain the type of Ada features that can be used		X	X	X				
4	To ensure programming uniformity, a style guide should be developed and used across all developer teams	X	X	X					
L18	**P-3C UPDATE IV Ada DEVELOPMENT**								
1	Ada code requires more up-front time and effort, and the learning curve is slower			X					
2	Increased facilities and memory are required to accommodate Ada code		X	X		X	X		

Figure I-1. Lessons Learned Matrix

Appendix Reference	Lesson	Standards and Policy	Project Management	Development Process	Corporate Knowledge and Software Development Training	Resources and Facilities	Support Environment and Tools	Reuse	Project Costs
3	Some software development tools are immature and have not been proven for many applications				X		X	X		
4	Tailoring of DOD software development standards must be addressed to accommodate Ada-unique capabilities	X	X	X						
5	Although the adoption of Ada was envisioned to enhance the software development process, use of Ada does not guarantee sound software engineering practice		X	X	X	X				
6	Strict configuration management and control are required to enforce discipline to counter complexity-induced confusion	X	X	X	X	X				
I.19	**STANDARD FINANCIAL SYSTEM REDESIGN**									
1	The available pool of developers skilled in Ada is limited		X	X	X	X				
2	Few compilers and support tools are available to support information systems development in the IBM environment that use Ada		X					X		
3	Systems developed in Ada may be more maintainable than those written in COBOL		X		X				X	
4	In principle, portability is ensured by developing code in Ada; however, in practice, portability is limited		X		X				X	

I.1 STRATCOM—COMPUTER CENTER, OFFUTT AIR FORCE BASE

STRATCOM—Computer Center has been using Ada since 1989 in many programs and projects. The center has built small systems (3,000 lines of code) and some medium systems (over 50,000 lines of code). Lessons learned from this work, which is mainly in intelligent databases, are described below.

Lesson 1: *Visualize Ada as supporting an engineering approach to software.*

Lesson 2: *Do not anticipate that use of Ada will decrease development time, particularly on the first few projects.*

Lesson 3: *Try to limit the number of variables in the development process (e.g., beta copies of software, new tools) on the first project.* Using too many variables will make it difficult to trace problems.

Lesson 4: *Do not blame all problems on the software or Ada.* Ada will often be blamed for anything and everything (whether it deserves the blame or not). Management often attributes to Ada other problems encountered in the use of Graphical User Interfaces (GUIs) or Database Management Systems (DBMSs).

Lesson 5: *Select a small doable project for the first Ada effort.*

Lesson 6: *Use tools.* The days of the "all-I-need-is-a-compiler" are over. Software engineering is a complicated business. Tools can be a "force multiplier" that helps the engineer boost productivity, establish consistency across software modules, and enhance understanding. Use of flowcharts is no longer sufficient.

Lesson 7: *Ensure that management understands that tools support software engineering, not programming.* Management should move from managing just programmers to helping engineers engineer software solutions.

Lesson 8: *Ensure that management focuses on the engineering aspects of Ada, not on Ada as a programming language.* As a language, Ada appears to be complicated, but Ada's intricacies support sound software engineering principles. Viewed in isolation, Ada is often seen as overwhelming. In the perspective of large systems software development, however, the features of the language take on a much more valuable meaning.

Lesson 9: *Use a well-defined, repeatable process and methods for developing and maintaining software.*

Lesson 10: *Implement an active training program in both Ada and software engineering.*

Lesson 11: *Ensure that adequate resources (e.g., hardware capacity, disk space, tools) are available.*

Lesson 12: *Have adequate configuration management.* A well-structured approach to development of a large project will require several program units, especially in an object-oriented environment. Proper management of these units is crucial when a multiperson team is updating the modules.

Lesson 13: *Assess software engineering capabilities before the project gets under way.* Management should know what the organization's weaknesses are and how they will affect an intense Ada or software engineering project.

Lesson 14: *Do not expect to transform maintenance personnel into developers overnight.*

Lesson 15: *Do not field products before they are ready.* Management tends to scrutinize Ada projects more closely than other projects (perhaps because it expects them to fail). This scrutiny often leads to delivering products before they are fully tested.

Lesson 16: *Tailor the Data Item Descriptions (DIDs) from Military Standard (MIL-STD)-2167A.*

Lesson 17: *Do not neglect documentation.* Well-written Ada is a must, but the code is not totally self-documenting. Requirements, design, and other life-cycle management documents also are needed to give the total development picture. You must provide a vehicle to update documentation (e.g., flowcharts, data flow diagrams) that is detached from the code as the code changes.

Lesson 18: *Use Commercial-Off-The-Shelf (COTS) products as well as reuse repositories, where available.* Bindings to existing software products should be written, tested, and then reused. Avoid reinventing what has been done before. A little research into existing software may save time in the long run.

Lesson 19: *Beware of the software part received from outside repositories.* Many parts contain useless code. Real applications need code that is done well and rigorously tested.

Lesson 20: *Do not make spot fixes without proper change control.*

Lesson 21: *Have a test strategy that includes regression testing.*

Lesson 22: *Have a thorough integration plan.*

Lesson 23: *Ensure the Project Manager has authority to direct efforts and resources.*

Lesson 24: *Clearly define the role of the Program Manager.*

Lesson 25: *Ensure that the Project Manager of an Ada project is literate in software engineering and Ada.*

Lesson 26: *Establish a system architecture (e.g., open system, client-server) and stick with it.*

Lesson 27: *Apply configuration management and Quality Assurance (QA) to all parts of the system (e.g., requirements, design, Ada code, database code, COTS products) regardless of the language.*

Lesson 28: *Do not flood the team with untrained people and expect them to get trained "free" on the job.*

Lesson 29: *Ensure that upper management and configuration management personnel understand that they do not need to know how to program in Ada to manage projects written in Ada.* To be effective, however, personnel does need to understand a little about Ada and software engineering.

Lesson 30: *Provide periodic updates to non-Ada groups outside the project.* People not involved in Ada projects tend to ignore the Ada projects.

I.2 WELLS FARGO NIKKO INVESTMENT ADVISORS

Wells Fargo Nikko Investment Advisors (WFNIA) is a joint venture between Wells Fargo (California) and Nikko Securities (Japan). WFNIA, a leader in the institutional investment management service industry, manages large sums of money at very low percentage rates. The computer software it uses is an important competitive differential. The paragraphs below summarize the lessons learned from WFNIA efforts.

Lesson 1: *To use Ada effectively, management at the highest level must be committed to Ada.*

Lesson 2: *Ada requires an up-front investment, patience to wait for the up-front investment, and patience to wait for the pay back.*

Lesson 3: *Some criticism, pessimism, and panic are almost inevitable.* Management must be able to see through the storm to the goal ahead.

Lesson 4: *Ada must be sold, and existing environments (and other languages) must be "un-sold."*

Lesson 5: *Support and commitment from other developers and users should grow over time as the language has time to prove itself.*

Lesson 6: *Tools and adequate hardware to run the tools are needed.*

Lesson 7: *Training is a must.* Ada-specific training and, more important, software engineering training are required.

Lesson 8: *A software repository foundation is vital with both externally developed and internally developed software.*

Lesson 9: *Ada can and should be used with other languages.* When Ada is mixed with other languages, the Ada portion should be approximately 85%.

I.3 B-2 AIRCREW TRAINING DEVICES

Aviation Week & Space Technology reports:

> The Air Force/contractor team is developing the B-2 simulators concurrently with the aircraft in a program that stresses close ties between aircraft and simulator personnel and ease of modification of the aircraft.
>
> From the training perspective alone, the program presents three major challenges:
>
> - To develop the simulators and aircraft concurrently
> - To have the simulators ready for use before the first aircraft
> - To ensure the simulators match the current aircraft, but are designed to be updated easily.

(*Aviation Week & Space Technology*, 20 August 1990, pages 34-42)

Stealth Training

The B-2 is the penetrating bomber of the future. The aircraft incorporates low-observable or "stealth" technologies to reduce its detectable signature in a wide array of sensor spectra.

Training the aircrews to effectively fly and apply a stealth aircraft presents unique challenges. To solve those challenges, the U.S. Air Force turned to CAE-Link, just as it did earlier for the F-117A stealth fighter/bomber and for the SR-71 Blackbird Program.

The Aircrew Training Devices (ATDs) being developed by CAE-Link for the B-2 Advanced Technology Bomber total training system "provide fidelity and training capability beyond any device yet developed," according to the Air Force ATD Program Manager.

The B-2 ATD program has taken a revolutionary approach to training concurrency, and the simulators being developed represent the largest real-time simulation application of the Ada programming language.

Under a competitively won contract, CAE-Link is producing three Weapon System Trainers (WSTs), two Mission Trainers (MTs) and a System Support Center (SSC). Northrop, the aircraft prime manufacturer, has overall responsibility for aircrew and maintenance training.

The Electronic Combat Environment for the B-2 ATDs provides a high-density environment of up to 12,000 threats per mission, selected from a threat library of 800 types such as radar emitters, surface-to-air missiles, antiaircraft, and other aircraft. Of the 12,000 threats, up to 2,000 may be located in the 50,000-square-mile mission area, and 256 may be active within the radar horizon of the aircraft. In effect, B-2 aircrews will be able to fly simulated training missions anywhere in the world against any contemporary mission environment.

This high-fidelity, real-time threat environment features dynamic reaction, weapon deployment with realistic missile flyouts, terrain occulting, and networking effects. In addition to the automatic reaction to the flight of the B-2 through the environment, commands are accepted in real time from the instructional system to change the operation of threats or to control the overall execution of the simulation. Examples of the commands accepted are as follows:

- Move, copy, add, delete threats
- Set threats to specific modes (search, track, launch, off)
- Inhibit or activate threats
- Inhibit or initiate weapon launch
- Initiate airborne interceptor attacks.

The real-time threat environment is supported by an off-line threat database that includes location, operational tactics, networking, signal parameters, weapon characteristics, moving platform attributes, and Electronic Counter-Countermeasures (ECCM)

capabilities. The threat database uses Government-provided standard intelligence tapes as a primary source of input data. This facilitates automation of the scenario-generation process by providing an automatic threat laydown with global coverage, which is suitable for mission rehearsal. The tape update capability also provides a rapid and convenient means of maintaining currency with the latest data available from the intelligence community. A full-screen edit capability is provided to allow manual creation, tailoring, and verification of threat data.

The B-2 computational system is by far the largest real-time simulation application of Ada to date. It is capable of handling 60 Million Instructions Per Second (MIPS). By comparison, the B-2 simulator has seven times the computing capacity of a B-52 WST.

The B-2 ATD system has more than 1.7 million lines of Ada code. In addition, the simulator uses 1.7 million of the 2 million lines of aircraft code to operate the Aircraft Control Unit Emulators (ACUEs). The emulators, commercial versions of the Paramax aircraft onboard computers, have slightly enhanced capability and memory. (Software for the B-2 aircraft is not written in Ada.)

Concurrent Computer Corporation is supplying CAE-Link with Model 3280MPS superminicomputers with 19 processors per simulator. The 3280s are connected by high-speed distributed bus links so that they can share memory, processors, and other resources—more than 60 times faster than a Local Area Network (LAN) connection would permit. Initially, only one-half the computational system will be used with 50% as spare; the architecture also permits 100% expansion.

Lesson 1: *In the early Design Phase, capacity and performance estimates should target resources that provide at least a factor of 2 reserve.* Such reserve capacity will limit the risk of contention's developing later.

When the decision was made in 1984 to use Ada in the B-2 simulators, the Ada language had only recently been frozen and a validated Ada compiler did not exist.

Lesson 2: *When COTS language development products are to be used, project staff should expend as much effort as possible to determine which products best meet project needs.* Technical, management, and business-sensitive issues and concerns need to be identified and evaluated, and contingencies need to be developed where problems persist.

Specially trained CAE-Link Ada developers, working as a team with experts provided by the Government and the Software Engineering Institute (SEI), wrote the software. CAE-Link brought in experts from Carnegie-Mellon University and others to validate the system independently.

Initially, it was difficult to learn how to apply Ada. The B-2 ATD, however, is now reaping many of the prized benefits of the DOD's new standard language. The B-2 training program has a proven set of integrated Ada-based tools and a disciplined, repeatable process that fully supports all life-cycle phases. The investment in committing to Ada is paying for itself again and again in the modification process.

Lesson 3: ***If well-thought-out engineering and management plans are implemented with enough resources, the project will be successful. Moreover, a proven methodology can be used repetitively to evolve the current and new systems, based on the low-risk reused assets.***

The B-2 Ada tool set includes a specially developed Ada-based configuration management tool and a closely integrated load-build tool. Both tools were written in Ada and meet CAE-Link's standard policies. These tools are mature today; however, process improvement is considered a normal part of the continuing CAE-Link process. "Our attitude," says CAE-Link B-2 Program Manager Keith Hickling, "is that improvement is always possible." For example, in the early stages of the program it took nearly 6 months to train an Ada engineer; today, however, the training time is closer to 6 weeks, and the goal is to reduce it to only 3 weeks.

"We are also continually looking at ways to improve engineering productivity," Hickling points out. Software metrics are key. "Metrics help us isolate the right process areas to improve." Improvements are then planned as part of the normal engineering change process, resulting in a process that continually gets better. "We are training our engineers faster, they are becoming productive sooner, and we are increasing our reusable Ada software. This reduces cost of modifications for the B-2 ATD and benefits the company and the customer."

The B-2 ATD program has proven that the benefits of Ada are genuine, "but only," Hickling notes, "if the front-end investment is made in a sound software process, tool-set, and training."

I.4 BOEING MILITARY AIRCRAFT (WICHITA, KANSAS)

Boeing is involved in many military and commercial aircraft ventures. Lessons learned from these ventures are related to the engineering support area, which supports the software tools and research needs for the aircraft development programs. The paragraphs below summarize lessons learned as they relate to:

- Portability (Lessons 1 and 2)
- Separate Compilation (Lesson 3)
- Readability (Lesson 4).

I.4.1 Portability

Lesson 1: *Careful technical management of the development of code directives, guidelines, and formulations enables project staff to maximize the amount of fully portable code that is produced.* The portability of Ada programs between implementation is now good and has improved considerably since the initial releases of Ada implementations. What is significant relative to other languages is that the class of problems that can be written without requiring implementation-dependent code is larger than in most other languages. This class includes tasking and time-dependent programs.

Lesson 2: *The availability of a large suite of formal validation tests and use of an independent testing organization help ensure the basic qualifications of vendor-supplied products.* Use of these tests and an independent team do not replace project staff's detailed analysis and testing, but they do help eliminate marginal products. This validation process will become more important as we move to Ada 94 and other object-based languages (e.g., C++).

The Ada Compiler Validation Capability (ACVC) tests and the Language Control Board have been fairly successful in avoiding the creation of incompatible versions of Ada. This reduction in the number of incompatible versions helps promote portability. Some problems still exist in this area. For example, some implementations do not support preemptive priority scheduling, presumably because the ACVC tests do not test for it. The situation in Ada, however, is much better than in many other languages (e.g., C, C++, and FORTRAN).

I.4.2 Separate Compilation

Ada provides for the separate compilation of units and will enforce strong typing between separately compiled units. This is a significant advance over the independent compilation of other languages in which type checking is often lost between units (or deferred until run time). A compilation system can provide an automatic recompilation facility to prevent dependent units from becoming out of date. Although this facility is not available on all implementations, the checking and reporting on errors when obsolete units are encountered is universal and very helpful.

Lesson 3: *Lessons learned with respect to portability (see Lessons 1 and 2) also apply to separate compilation.* An Ada programmer quickly learns to take for granted the separate compilation and the type-safe checking it provides and can easily forget how difficult it is to track down errors when using "independent" compilation in other languages. Neither the fact that Ada's separate compilation facility becomes invisible to users nor the fact that it was one of the explicit goals of the language the achievement of which is not surprising should detract from the value that the facility provides.

I.4.3 Readability

Lesson 4: *Controlled use of Ada language constructs results in uniform and minimally complex code, thus maximizing readability.* It is possible to write some very obscure code in Ada by using overloading and derived types and multiple levels of generic units. In general, however, the language permits most programming tasks to be coded in a fairly straightforward way. This language power facilitates development of readable programs. Programmers do not often have to "code around" limitations in the language (or use vendor-specific extensions) as is too often necessary in other languages (e.g., doing dynamic allocation in FORTRAN, operating on unconstrained types in standard Pascal, writing functions in COBOL).

I.5 COULTER ELECTRONICS: Ada FOR CYTOMETRY

Coulter Electronics develops machines to analyze blood. The paragraphs below summarize lessons learned on three small Ada projects that run on a Personal Computer (PC) platform.

Lesson 1: *Look at the language and the constructs to be used and decide on an environment.*

Lesson 2: *Evaluate your needs and then evaluate the compilers that run on your particular platform.*

Lesson 3: *Look at external software programs that have to work with your particular program.*

Lesson 4: *Ensure that the compiler has a method for accessing external hardware interfaces if the project equipment has such interfaces.*

Lesson 5: *Remember that "optimization" should minimize the code size not just remove "dead code."*

Lesson 6: *Recognize that reuse can be a major factor in code development if looked at from the beginning.*

I.6 AN/UYS-2A PROJECT

The AN/UYS-2A, which is under the direction of the Naval Sea Systems Command (NAVSEA) PMO-428, is a programmable, data flow, high-throughput, modular Navy standard signal processor. The AN/UYS-2A consists of a family of signal processors that meets the diverse environmental requirements of ship, shore, submarine, and aircraft Anti-Submarine Warfare (ASW) platforms. Because of its design, the AN/UYS-2A is easier to program and costs less over the system life cycle than the previous system. The

AN/UYS-2A is a Standard Embedded Computer Resource (SECR) and is not designed to meet or counter any specific threat on a stand-alone basis.

The basic AN/UYS-2A is composed of different combinations of seven Functional Elements (FEs): Arithmetic Processors (APs), Input Signal Conditioner (ISC), Global Memories (GMs), Input/Output Processors (IOPs), a Command Program Processor (CPP), a Scheduler (SCH), and a Data Transfer Network (DTN). Additional functional elements may be added to the basic AN/UYS-2A processing capabilities. These elements can be matched to each weapon system's requirements by selecting the combination of APs, GMs, IOPs, and ISCs that best satisfy the requirements of the individual weapon system. The AN/UYS-2A is also modular at the logistics level. That is, each of the seven functional elements is built from a common set of format E Standard Electronic Module (SEM) cards. Although the terminology has changed from SEM to Digital Electronic Module (DEM), many documents still use the term SEM. The terms are interchangeable.

Lesson 1: *When selecting a host computer as the Application Development Facility, ensure the selected host computer supports Ada so that application developers do not need to purchase multiple hosts to develop application software.* The SEM B AN/UYS-2's CPP used an embedded AN/UYK-44(V) card set that ran the Navy's Compiler Monitor System-2 (CMS-2) language. Because the CMS-2 language software development tools reside on the VAX environment, the decision was made to select the VAX 11/780 as the ADF host computer. The SEM B AN/UYS-2's CPP uses a Motorola 68020 architecture and was required to use Ada as the AN/UYS-2 Command Program language. Unfortunately, an Ada M68020 cross-compiler was not available for the VAX 11/780; therefore, the decision was made to use a Telesoft Ada compiler environment running on a Sun platform.

Lesson 2: *When selecting Ada products, ensure that the Ada vendor can provide a full spectrum of products (i.e., hosts, cross compilers, Run-time Kernels). Avoid using multiple vendors when possible.* The Ada environment selected was a combination of Telesoft's compiler and Ready's Run-Time Ada (RTAda) extensions. The chronological sequence of events was as follows:

- RTAda was purchased from Ready Systems.
- Ready Systems contracted with Telesoft for the Ada compiler and run-time interface code.
- Ready Systems modified the run-time code to support the Ada Run-Time Executive (ARTX).

- Ready Systems integrated, sold, and maintained the RTAda product for AN/UYS-2A.
- The internal contract agreement between Ready Systems and Telesoft expired on 31 December 1990.
- Ready Systems stopped selling and supporting the RTAda product.
- The AN/UYS-2 customer could not purchase the RTAda product or services.
- AT&T contracted with Telesoft to develop a Telesoft Run-Time Environment (TeleAdaExec).

Lesson 3: *Select a well-established Ada vendor who demonstrates willingness to help software developers move code to new versions of their compilers.* The Telesoft compiler was upgraded several times during the SEM B AN/UYS-2A development effort. Version 1.3 was upgraded to 4.1A and 4.1A to 4.1C. Although the modifications enhanced the compiler by providing more complete data and path checking and greater code efficiency, they resulted in additional compiler restrictions. Consequently, some Command Program Ada code had to be rewritten so that it would be compatible with the newer compiler version.

Lesson 4: *Establish a close working relationship with the Ada vendor and define project needs as early as possible. Plan Ada upgrades in a systematic and controlled manner.* On the AN/UYS-2A project, special efforts were made in working with Telesoft to determine the direction of future compiler upgrades. Project management and staff also tried to communicate to Telesoft the evolving program needs and concerns.

I.7 Ada EXPERIENCE AT THE NAVAL RESEARCH AND DEVELOPMENT CENTER

In 1988, the support staff and the contractor of the Naval Research and Development (NRaD) Center Code 924 began the transition from use of CMS-2 and its traditional software architecture to Ada and an object-based design philosophy. This change was prompted by the decline of the then-current product line into a caretaker status, without funds to match the magnitude of knowledge needed to protect Government interests.

The situation presented a rare opportunity both to accept the challenge of transitioning from CMS-2 to Ada and to document that experience. Contracting was being performed under a time-and-materials contract, thereby simplifying statistical measurements because such contracts are monitored on a labor-hour basis. The new software products to be implemented in Ada included CMS-2 source analysis tools; data reduction programs; and real-time, interactive PC-based products. It should be noted that comparing the statistical

numbers of one project to another is difficult because there are so many variables. It is better to compare baseline to baseline within a given project. Even then there can be distortions.

The paragraphs below summarize the lessons learned from a management perspective rather than from a programmer's perspective. Programmers would be more interested in language-specific application lessons.

Lesson 1: *The transition to object-based design and use of Ada enhanced productivity. It decreased integration time because there were fewer errors and less need to rework code.* The older CMS-2 software engineering process as applied to systems programming in NRaD yielded a productivity rate of 250 Source Lines Of Code Per Staff Month (SLOC/SM). Transitioning to Ada and adopting an object-based approach increased the productivity rate by 24% (i.e., 310 SLOC/SM). The expressive power of Ada also increased productivity. Function Point (FP) productivity tables show that an FP implemented in CMS-2 requires approximately 105 SLOC, whereas Ada only requires 70 SLOC. The productivity advantages became apparent to the NRaD support staff as a result of Ada's support of abstraction and encapsulation and the rapidity with which the Integration and Test Phase of a given implementation was completed.

An analysis of the errors encountered during the production process showed a 21% reduction in errors. Although industry samplings show even greater reduction (i.e., 24%), further analysis is required to ensure that the basis for comparison is consistent. Code 924 staff believe that their figure represents a more arduous test process. Factors contributing to this improvement are the level of error checking in Ada compilers, use of Ada features that support a self-documenting style, and implementation of information-hiding concepts that reduce the side effects found from the use of traditional common stores.

Lesson 2: *The use of automated tools and Ada have enhanced our ability to maintain developed products and their documentation.* Maintainability has been greatly enhanced. Use of a software engineering process that combines the use of Ada as a Program Design Language (PDL) and emphasis on code readability has allowed errors to be corrected rapidly. Development and use of an Ada Reuse Library Browser (ARLB) further enhanced maintainability. The ARLB allows the programmer to rapidly traverse call trees and WITH dependencies to focus on individual package bodies where source and design representation modifications are made interactively. The ARLB, supported by disciplined programming standards, has led to automated design document production derived from the Ada source library.

Lesson 3: *Project management should expect that at least 50% of the development time will be spent in the Requirements Analysis and Design Phases.* Deriving the objects and

their associated operations into Ada package specifications is an iterative process requiring considerable time and interaction among the lead designers. Elaborating a design to implement those objects and operations, using an Ada PDL, into the Ada package bodies represents an additional up-front investment. Patience was required because the overall design process consumed 50% of the implementation time. After coding began, however, it progressed rapidly and integration occurred quickly with fewer errors. The overall schedule (in months) seemed to be the same as that for a CMS-2 program; however, a smaller staff was required. We are not sure whether increasing the number of staff members would shorten the schedule.

Lesson 4: ***Attitude is a key factor in transitioning engineering personnel to modern software engineering and Ada.*** Success will only come from a well-motivated team that is committed to the tool, technology, and project.

Training is critical to preventing the application of Ada in the context of traditional CMS-2 design disciplines. The Ada language was designed to support more modern software engineering approaches and should be used in that context. The critical paradigm shift is one from the classical hierarchy of processes to one of object orientation. For most programmers, this shift can be achieved in 4 to 9 months through a combination of classroom training and on-the-job experience. New college graduates adapt quickly. Many of the older CMS-2 programmers may never make the transition. Older programmers should not be forced into a position of resistance to change. To be successful, the job must be in the hands of believers.

Traditional CMS-2 systems have been built with a specific computer in mind. The software was dependent on the machine-sensitive constructs of the language of implementation and the service calls of the chosen executive. Dialect difference between implementations of purported standard languages and operating systems have limited the market of the implemented systems to hardware supported by the compiler or operating system vendor. With Ada's rigorous standards, code has benefited from the ability to draw software components from a common library and use compilers of multiple vendors to place its products on a variety of target hardware—an important consideration in an era of migration from gray boxes to the richer mix of architectures in the commercial arena.

I.8 TACTICAL AIRCRAFT MISSION PLANNING SYSTEM

The Tactical Aircraft Mission Planning System (TAMPS) is hosted on the Navy's standard Desktop Computer (DTC-2). With the release of the New Tactical Advanced Computer (TAC-3) as an upgrade replacement for the DTC-2, Naval Air Warfare Center, Aircraft Division Warminster (NAWC-AD WAR) is tasked to evaluate the TAMPS software portability to the TAC-3 platform.

The subsections below identify problems associated with porting TAMPS software from the DTC-2 to the TAC-3 platform and illustrate the magnitude of each problem.

I.8.1 TAMPS TAC-3 Hardware and Software Configuration

The TAC-3 hardware suite, delivered to the NAWC TAMPS laboratory on 13 July 1992, consisted of the Hewlett-Packard (HP) 9000 Series 750 with 128 megabytes (MB) of memory, two 1.2-gigabyte (GB) disk drives, one 4-millimeter (mm) Digital Audio Tape (DAT) drive, and one monitor. The TAC-3 software included the HP-UX Operating System, the Irvine Compiler Corporation (ICC) Ada compiler, an HP-UX FORTRAN compiler, and an HP-UX C compiler. This system configuration is only sufficient to recompile and to evaluate TAMPS code portability. A complete hardware suite is required to evaluate TAMPS executability after all compilation errors have been resolved.

I.8.2 TAMPS Evaluation Results

NAWC used the TAMPS 5.0x3 source code to evaluate its portability from the DTC-2 to the TAC-3 platform. The evaluation task was divided into the following areas: Hardware, Operating System, Compiler and Support Software, and Peripheral and Device Driver. The subsections below list problems uncovered for this task for each area and provide impact assessments.

I.8.2.1 TAMPS Hardware Assessment

Because the internal data representation of the two machines is the same, the TAMPS databases can be transferred to the TAC-3 hardware and used without any conversion. NAWC wrote a routine to read or write data onto a file on the DTC-2 and used the same routine to read the data back onto the TAC-3. The results showed that the internal data representation on both systems was the same. BTG, Inc. (i.e., the TAC-3 technical support contractor) confirmed our results.

The TAC-3 graphics processors support two independent workstations and a DBA station with X11R4 libraries, which will satisfy TAMPS requirements. TAMPS software, however, needs to be tested on the TAC-3 hardware to confirm that all TAMPS graphics requirements will run without further software modifications.

I.8.2.2 TAMPS Software Assessment

After the TAC-3 hardware suite was set up, NAWC began TAMPS software assessment. The HP-UX Operating System (System V), ICC Ada compiler, HP-UX FORTRAN compiler, HP-UX C compiler, and X11R4 libraries were used to assess TAMPS code.

The Ada, C, and FORTRAN compilers were installed and verified. Then the required libraries were created as indicated in TAMPS makefiles. Because the "makefile" commands on the two systems were different, new TAMPS makefiles were written to

recompile TAMPS on HP-UX.

The HP Window Manager (Vuewm) and X11R4 libraries supplied with HP-UX were tested by running standard X-based applications. In addition, the manual pages for the Vuewm were compared with those of the Motif Window Manager for discrepancies. A list of system calls in TAMPS was gathered by the UNIX "grep" command. The parameters and usage of the system calls were compared to determine the differences. Ada, FORTRAN, and C files were recompiled, and error listings were examined to determine the problems and solutions.

Because of the incompatibilities between the ICC Ada compiler and the Sun Ada compiler, NAWC is acquiring the Alsys Ada compiler to perform another TAMPS Ada code assessment at the NAWC laboratory.

Lesson 1: *Before selecting vendor products, it is important to test them extensively to ensure that they meet a project's specific needs.*

I.8.2.3 Operating System

The operating environment, Vuewm, is an X11 window manager based upon the Motif Window Manager (mwm, version 1.1). Vuewm is an integral part of the HP Visual User Environment (HP VUE). It communicates with and facilitates access to the other components in the environment. Vuewm provides the same window management and limited session management functionality as mwm. It allows the user to control window size, position, state (iconic or normal), input focus ownership, and the like. TAMPS will be able to run in this environment with little problem.

Most of the system calls used in TAMPS (DTC-2/BSD Operating System) are compatible with those in the TAC-3 (System V Operating System) except for the following types of problems:

- Different constants
- Function names
- Unsupported asynchronous Input/Output (I/O)
- System calls that are not in TAC-3.

All shared memory calls are compatible between the DTC-2 and the TAC-3 system. This area needs to be tested when all of the other problems are resolved to confirm that TAMPS will run without further software modifications.

I.8.2.4 Compiler and Support Software

Of the 55 C files within TAMPS code, 44 files were compiled without errors and 11 files (or 20%) could not be compiled because of the following types of problems:

- Different library functions
- The nonportable code for system functions in the Computer Software Configuration Items (CSCIs).

For all C implementations, new code had to be generated to handle the library functions and nonportable code problems.

All TAMPS FORTRAN code has been recompiled in the TAC-3 system with the HP-UX FORTRAN compiler. Of the 3,453 FORTRAN files within TAMPS code, 3,392 files (98%) were compiled without errors and 61 files (or 2%) could not be compiled. Problems found while recompiling TAMPS FORTRAN code are as follows:

- *Overlapping data initializations.* The FORTRAN compiler does not allow a variable to be initialized more than once in a data statement.

- *Error due to the alignment in the common block.* Integer variable must start at an odd address.

- *Explicit definition of format statement needed.*

- *Character string referenced out of range.* A character string is defined with a length N and later used with a length of N+m.

- *Nonlogical expression in IF/DO WHILE statement.* An integer variable is used as a logical variable.

- *Nonpositive label.* A label of zero is used in TAMPS code. A label must be within the range of 1 to 99999.

- *Argument with the same name as INTRINSIC function.* TAMPS code uses the INTRINSIC function "FLOAT" as one of the arguments in a parameter to a subroutine.

- *Adjustable array in common block.* In a few places, TAMPS code defined an array in a common block as:

  - Integer length
  - Common XXX / Array YYY(LENGTH) /.

This FORTRAN compiler cannot figure out the size of LENGTH because it is declared but undefined at this time. Therefore, it cannot declare the array YYY.

These FORTRAN problems have a minimal impact on TAMPS code.

All TAMPS Ada code has been recompiled in the TAC-3 system with the Ada compiler from ICC. A few of the problems with TAMPS Ada code were serious because of the incompatibility between the DTC-2 Sun Ada compiler and the TAC-3 ICC Ada compiler. Independent research had shown that many users were having trouble with the ICC product. Two basic modifications were required before Ada code could be compiled with the ICC Ada compiler. First, the ICC Ada compiler treated "subtype integer" in the same way as it did "standard.integer". Therefore, the basic integer types in the "basic_data_types_pkg.ada" package were redefined. Second, the "LANGUAGE" package was Sun Ada compiler's unique package, and all pragma statements referencing the LANGUAGE package need to be commented out.

After completion of the above basic modifications, 2,969 Ada source files remained within TAMPS code: 352 files that were compiled without errors and 2,617 files (or 88%) that could not be compiled. These errors result from the different implementations of the two compilers.

Lesson 2: ***Many details in the implementation process are not controlled by MIL-STD-1815A or the associated validation suite for the Ada language.*** Project staff should perform sufficiently detailed analysis of particular implementations so that they can correctly assess impact when changing configurations.

The following paragraphs list all general problems found while recompiling TAMPS Ada code:

- *Misalignment.* An integer field declaration in a record must lie in a word boundary.

- *Dynamic Generic Instantiation.* UNCHECKED_CONVERSION cannot be instantiated with dynamically sized type with the ICC Ada compiler.

- *Unsupported Machine_Code Package.* TAMPS uses inline expansion of low- level machine code provided by the Sun Ada compiler's "Machine_Code" package. The ICC compiler does not provide a Machine_Code package for the TAC-3 platform.

- *Unsupported ERRNO Package.* TAMPS uses the error package "ERRNO," which is specific to Sun's Ada compiler. This package is not provided with the ICC Ada compiler.

- *Unsupported System "+" Function.* Function "+" in Sun's system package does not comply with MIL-STD-1815A. This function is an extension provided by the Sun Ada compiler but not by the ICC Ada compiler.

- *Unsupported System.No_Addr Type.* The type "No_Addr" in Sun's system package does not comply with MIL-STD-1815A. This type is an extension provided by the Sun Ada compiler but not by the ICC compiler.

- *Calendar.Local_Time Package.* The "Local_Time" package within the Calendar Package does not comply with MIL-STD-1815A. TAMPS modifies a body part of the Calendar.Local_Time package and incorporates it into the standard Calendar package. Problems occurred when attempts were made to incorporate it into the Calendar package provided by the ICC compiler.

- *Disallowed Zero-Length Field in Record.* In TAMPS code, a field length of zero in a variant record is defined as null. The ICC compiler interprets it as a missing field and indicates it as an error.

- *Unincorporated Parent Package Name.* When a function is defined in a separate procedure, the ICC Ada compiler requires the parent package name must be "with" into the function code. The Sun Ada compiler does not have this requirement.

- *Unsupported VADS Configuration Package.* TAMPS uses a Verdix Ada Development System (VADS) Configuration Package Specification for Sun4 BSD UNIX. This package specification defines and describes the components that the user must provide to configure the VADS self-hosted Run-Time Environment (RTE) for a user application program. Users have the choice of using the Sun-supplied memory allocation packages or implementing their own algorithms. MDMSC should try to avoid all machine or compiler dependencies in the TAMPS code.

The problems associated with the incompatibilities of the two compilers required NAWC to use another vendor product, (i.e., the Alsys Ada compiler) to reduce the impacts on the TAMPS Ada code.

Lesson 3: *Staff should do up-front technical evaluations.*

Other areas of concern related to porting projects include the following:

- File structure and handling systems that are in use
- Peripheral and device drivers movement
- Special application software packages.

I.9 ADVANCED FIELD ARTILLERY TACTICAL DATA SYSTEM

The Advanced Field Artillery Tactical Data System (AFATDS) is a system of computers, printers, displays, and software that helps Army commanders plan, direct, and control artillery fire in combat situations. AFATDS was intended to replace the former Tactical Fire Direction (TACFIRE) system.

AFATDS was a concept evaluation effort that began in May 1984 with Magnavox Electronic Systems as the prime contractor. The paragraphs below summarize the lessons learned during this effort.

Lesson 1: *Anticipate trouble with the Ada development tools/environment, no matter who is supplying them or when you get them. Especially expect problems with the ability of the Ada Run-Time Executive to meet all of the project needs.* The Army had required Ada as the High Order Language (HOL). During the Source-Selection Phase, only three validated compilers were available, none of which could down-line load to a target processor that met the AFATDS-derived requirements. The language, methodology, and tools were new; the approach was to be "software first."

Lesson 2: *Budget for training. Be prepared for and include additional funds for training over a long period of time. Note that for this training to be most effective, it must be accomplished just before or during the development effort.* Magnavox recognized that real-time expertise in Ada did not exist and immediately went to the Ada community to establish a comprehensive, long-term Ada and software engineering training program. Magnavox also proceeded to hire selected consultants and subcontractors to handle specialty items (e.g., database design).

Lesson 3: *Anticipate that original estimates for support hardware and facilities will have to be revised. In this project, original estimates quadrupled for support hardware and facilities.* Magnavox also purchased multiple mainframe and workstation computing systems; however, these resources proved insufficient but were relatively easy to upgrade.

Lesson 4: *To accomplish the project successfully, ensure that both the contractor and Government teams are knowledgeable about and understand the rationale for all software-related topics.* At that time, none of the DOD policy standards had been updated (this is still true today in many cases), and very few people on the Government side understood their ramifications. The Army had taken a sound, long-term view when it awarded this contract, but early into implementation, the pressure of outside scrutiny

began to erode that resolve. This, coupled with limited understanding of Ada and its software engineering ramifications, caused serious disconnects to develop between the contractor and the Army acquisition team (e.g., "Where's the code?" syndrome).

Lesson 5: *Have the team develop a viable technical/management plan and adhere to it so that requirements and design can be implemented correctly. Although it will take longer to begin writing the actual code, it will be worth it because fewer design problems will be encountered during test and integration.* Some of the hardest work will be associated with trying to handle the external nay sayers.

Lesson 6: *Report major problems up the line as encountered.* Magnavox and the Army Program Office were never assertive in promoting their initiatives. Had they been, many of the external groups might not have felt compelled to investigate, and more time would have been available to resolve the technical problems. Others can benefit from lessons learned only if they are informed about them. Such publicity could have helped the AFATDS project and provided insight to other projects that were beginning.

Lesson 7: *Do not mistakenly blame software development for failure. Careful scrutiny of many projects frequently shows that things other than software development are responsible for failure.* For AFATDS, three formal General Accounting Office (GAO) evaluations were performed and reported on during 1986-87: GAO/NSIAD-86-184FS, GAO/NSIAD-86-212FS, and GAO/NSIAD-87-198BR. None of these reports identified Ada as a problem. Major impact items included the reduction in scope because of budget constraints, the changing of requirements to accommodate different equipment and software, and the Army's ability to manage this activity.

I.10 AN/BSY-2

The AN/BSY-2 Submarine Combat System (SCS) is the suite of hardware, software, and equipment that will be used on the Department of the Navy's (DON's) next-generation, attack-class submarine, the SSN-21. General Dynamics Electric Boat Division is building the first hull in this series, which will be ready in 1994.

Lesson 1: *When external schedule constraints exist, the level of planning and execution analysis becomes much more critical.* This was especially true for BSY-2 because of the estimated volume of software and separately defined hull completion dates. The AN/BSY-2 software is being developed under DOD-STD-2167A in an effort that has combined aspects of the Concept Evaluation, Demonstration and Validation (DEMVAL), and Full-Scale Development (FSD) Phases of the life cycle. Commencing in 1985, a draft set of DON-generated SCS requirements was used for the System Design Definition (SDD) activity. Leading up to FSD and contract award, the two successful bidders, IBM and General Electric, worked with the Navy team to solidify requirements, develop design approaches, analyze ongoing prototyping efforts, identify critical items, fine tune

the FSD Statement of Work (SOW), and generate three separate Source Lines of Code (SLOC) preliminary size estimates for the AN/BSY-2 System.

The other lessons learned on AN/BSY-2 fall into six distinct categories: contract, coordination, process, schedule, standards, and tools. Multiple lessons are presented for each of these areas. Note that the lessons do not apply exclusively to an Ada development and that they are presented randomly within each category (i.e., no attempt has been made to rank them).

Lesson 2: *Most of the "lessons learned" are related to the contract requirements.* The SOW should require regular reports on the status of all commercial products delivered as part of the system. This update should include information such as vendor, version number, performance statistics, licensing agreements, and plans for future modifications. In addition, when the same type of documentation is to be produced by multiple developers, implementation of a standardized style guide should be referred to in the SOW. Furthermore, a provision should be included to allow deliverables to be transmitted in an electronic format. On systems that have classified information, installation and use of encrypted links between developer sites should be mandatory.

To ensure that requirements flow down adequately, the prime contractor should be required to provide copies and/or updates of all subcontract agreements to the acquisition agency.

To be fully effective, software Quality Assurance (QA) should be totally independent and organized to avoid a double chain of command (i.e., having a development program in the place of corporate QA).

Identification, reporting, and close monitoring of available metrics should begin early in development. The level of detail should increase in tandem with advanced development. Metrics should be analyzed thoroughly, and results should be incorporated into quarterly program assessments. Progress or regression relative to the program plan baseline should be a key element in this assessment. Separate analyses conducted by DON for comparison purposes produced additional benefits for AN/BSY-2 when results of these analyses were shared with the developer.

To ensure that the metrics data received are comparable across all development teams, a uniform SLOC counting methodology must be defined and followed.

Lesson 3: *Coordination frequently receives the least attention although it is one of the more important efforts.* Early in the contract, direct lines of communication should be established among key participants: acquisition agency, developer, technical agency, Independent Verification and Validation (IV&V) agency, quality personnel, and COTS

software vendors. Such "shortcut" communiques result in more efficient problem identification and resolution, which have an overall positive effect on cost and schedule.

Informal networking among groups of like interest will increase the effectiveness of each group. Regularly scheduled communication tends to short-circuit problems while providing a broader perspective to participants. For example, AN/BSY-2 holds a monthly user group meeting to discuss problems, workarounds, and successes with the operating system. The vendor's active participation at these meetings has increased responsiveness to and visibility of AN/BSY-2 needs.

The prime contractor should maintain tight control of subcontractor efforts through weekly monitoring and quarterly audits. Furthermore, attendance at technical and working group meetings should be mandatory for all team members.

Lesson 4: *For large projects, it is mandatory that an adequately sized, qualified Technical Directive Authority (TDA) Oversight Group be established and function for the duration of the project.* Very early in development, the contractor should detail each process proposed for use in the program. These processes should be defined in approved, baselined documentation. DIDs need to include more stringent, detailed guidelines. Multidisciplinary contract agency representatives should then closely review each process in software development and in related areas (configuration management, QA, testing) for adequacy, consistency, and completeness. Contractor modifications to these processes should be presented during formal reviews and entered into the baseline document only upon approval.

A streamlined waiver request process should be established for reporting proposed contract deviations to language and/or contract requirements. Waiver packages should be initiated every 6 months, depending on program size and life span.

A comprehensive Ada training program should be developed to address application-specific requirements. This program should be capable of transitioning seasoned engineers yet flexible enough to instruct entry-level programmers.

Ada methodologies (e.g., exception handling) should be defined early in development. Partial tasking should be considered as an alternative for reducing rendezvous time. Establishing global error models well in advance of detailed design will result in a more robust system.

Ada guidelines and procedures should be established primarily by the program's resident Ada experts. These lessons learned should be provided in an Ada style guide as an appendix to the software Standards and Procedures Manual. For example, compilation dependencies can be reduced and debugging smoothed by avoiding subprogram nesting.

This think tank of Ada experts should also be convened to resolve complex, persistent, Ada design problems. For example, enhancement of time-critical processes can be effected through expert application of Rate Monotonic Scheduling techniques.

Lesson 5: *Software development planning and monitoring must be done from the onset of FSD and should take a phased approach (i.e., "build a little, test a little").* Ada software development schedules should allow for longer Requirements and Design Phases and shorter Test and Integration Phases. The schedule should contain Critical Design Reviews (CDRs) to correspond to the incrementally developed software. In addition, testing should use manageable units at phased steps with explicit success criteria.

The delivery schedule for software plans, standards, and procedures should show compressed early deliveries. Multiple early deliveries should accelerate establishment of a baseline. These planning documents should be baselined and under formal configuration control no later than at the close of the Preliminary Design Phase. Conversely, software requirements specifications should have fewer deliveries, a longer document review cycle, and a baseline before preliminary design.

Product Readiness Reviews (PRRs) should be held early in development. These reviews have a positive, cohesive effect and provide a close, systemwide look at processes, products, personnel, and facilities. Implementation of an action item system is key to PRR effectiveness.

The developer should identify critical-path software items (e.g., shared system services). Close management of this process should ensure early delivery and test of these functions.

Lesson 6: *Even the best-made plans require changes during execution.* AN/BSY-2 used DOD-STD-2167A for software development guidance. The intent of this standard, however, is to provide a software development superset from which extraneous requirements can be eliminated. AN/BSY-2 staff carefully tailored this standard, mindful that it is easier to provide relief from requirements than to "buy" them in later. The contracting office should remain open to negotiations on tailoring DOD standards as phases unfold, technology advances, and/or lessons are learned. As an example, support software documentation has been reduced from the full suite to design notebooks and operator or maintenance manuals.

As part of tailoring the standards, a cross-check should be performed against the SOW. Checking requirements in the SOW for potential ambiguities or even conflicts within the military standards may avoid costly rework during later phases.

Lesson 7: *For large efforts that are geographically dispersed, the goal should be to strive for commonality of development environment, tools, procedures, and product structure.* The contracting agency should require standardization of support tools across the program. Although the up-front cost is greater, long-term benefits gained from such commonality make it a worthwhile investment. Use of common tools allows problems to be identified and workarounds made only once and results entered into a shared electronic reporting system. In addition, data exchanges among development teams are less time-consuming and more efficient, thus reducing the risk of error.

For large projects, it is imperative that the configuration management system be capable of supporting rapid turnaround during the Integration and Test Phases. The system should provide configuration management of all software support tools as well as the development code. In addition, a version control process must be established and enforced by the prime contractor for these tools.

A common database should be established to electronically track requirements down through software requirements specifications and hardware unit specifications and, later, into test. Use of this method will enhance traceability and ensure flowdown of requirements. A common database should also be created to track connectivity of software interfaces. Consistency checks should be run for early detection of misaligned interfaces.

Commercial support tools may require modifications to handle large Ada developments, and non-Ada commercial code slated for incorporation into the product may create interface and performance problems. Additional time and resources should be factored into development plans to allow for these potential stumbling blocks. (Computer resources should also be supplemented to account for the increase in demand that traditionally occurs during Ada developments.)

Compiler benchmarks should be evaluated before compiler selection is finished. (Compilation time should be factored in as an additional consideration.) The developer should know the weaknesses as well as the strengths of the Ada constructs (e.g., link library sizes and nesting of generics) as used in the compiler and/or Ada Programming Support Environment [Ada PSE]. Binding approaches should be established and benchmarked early in the development.

Use of an Ada standards checking tool is highly recommended. Using a standards checker not only encourages production of high-quality code but also reduces staff efforts and enhances maintainability.

I.11 ADA LANGUAGE SYSTEM/NAVY

The Ada Language System/Navy (ALS/N) FSD program implements Ada for use with DON's standard embedded computers: AN/UYK-43(V), AN/UYK-44(V), and the P3I AN/AYK-14(V). Since January 1989, DON has mandated the use of ALS/N as the first-line support software consideration for the DON standard processors. Although ALS/N is a support software effort, it also is a large software-based systems development effort. The ALS/N development project has produced more than 1 million lines of Ada code that also support DOD-STD-2167A documentation.

The DON Ada Standard Embedded Composite Resource (SECR) effort began in the early 1980s and closely monitored the other Service efforts, such as the Army Ada Language System (ALS) effort and the Air Force Ada Integrated Environment (AIE) effort. The DON goals were to avoid reinventing the wheel and to maximize the benefits of the Ada reuse and portability concepts for developing support software. In 1984, DON opted to establish the baseline with the Army ALS and proceeded to develop specific SECR-retargeted compilers and tools.

Lesson 1: *For DON SECR applications, top priority must be given to the real-time performance of the generated code. Performance requirements must be formally specified, and performance capabilities must be tested before product acceptance and deployment.* Because of the number and severity of the problems encountered, the Army paid little attention to performance issues for the support environment and the targeted real-time environment.

Lesson 2: *Although actual software code production is only a relatively small portion of the total life cycle, it is critical to have a reasonable level of performance within the tool set. At a minimum, the tool set must meet both programmer functional and configuration management needs.* The Army ALS tool set had been implemented in Ada but operated on the VAX/VMS host environment through an additional layer called the Kernel Ada Programming Support Environment (KAPSE). This arrangement made tool performance unacceptably slow. The Navy, therefore, redirected the contractor to eliminate the KAPSE requirement.

Lesson 3: *Each development effort should be managed under the assumption that there will be a formal production delivery to DON and a separate DON-controlled Post-Deployment Phase.* To ensure continuous development oversight, DON laboratory personnel were provided to fa tate the transition to life-cycle support.

Lesson 4: *Requirements must be understood, and both formal and informal checks on the progress to meet these goals must be conducted throughout development.* The Air Force used an independent test team in this effort and spent 15% of the budget on it. This team performed Technical Directive Authority (TDA)-type testing that included full

knowledge and understanding of the product internals. Concurrently, a separate IV&V agent performed "black box" testing to evaluate formally the specified requirements. Expenditures for this support were approximately 5% of the total budget.

Lesson 5: *Because post-deployment support will be DON's responsibility, it is critical to build an adequate in-house team that is thoroughly familiar with the product before acceptance.* The ALS/N development has actively funded various Navy laboratories (e.g., Naval Surface Weapons Center [NSWC], Naval Avionics Center [NAC], Naval Undersea Command [NUSC], Naval Air Development Center [NADC], and Naval Ocean Systems Center [NOSC]) to participate in the program and also involved the Navy's life-cycle agent (i.e., Fleet Combat Direction System Support Activity [FCDSSA], San Diego).

Lesson 6: *Lack of full program funding commitment and support will have a negative impact on development plans. Be prepared to either alter the course of and/or extend delivery schedules. Always try to maintain the best possible product quality and maximize life-cycle supportability within the program constraints.* The vagaries of year-to-year funding support tend to disrupt large undertakings that involve many elements such as laboratories, prime contractors, subcontractors, IV&V, and independent test organizations. All parties have to be motivated, good informal communication mechanisms must be in place, and all development efforts must be carried out according to an agreed-to plan that can accommodate a certain degree of flexibility.

Lesson 7: *Producing a high-quality software-based product that meets its specified requirements is a difficult task.* ALS/N provides a software means to upgrade deployed SECR processor-based systems indefinitely. ALS/N also can be considered as the front-line consideration for new systems developments because DON has 100% ownership or change control rights. Many U.S. commercial companies provide Ada compiler technology. Investment costs for those technologies that have been commercially successful are consistent with DON expenditures for ALS/N. However, few of these commercial Ada technologies specifically addressed real-time performance to the degree of ALS/N capabilities, which is required for Mission-Critical Computer Resources (MCCR) applications. In fact, two out of every three DON dollars have been spent on DON standard RTE needs. The ALS/N FSD program has produced compilers and run-time operating systems that will meet many of the performance requirements as specified.

Lesson 8: *No product is truly exercised and tested until it reaches the target user community. It is best to phase systems into deployment through beta testing and friendly users before public release.* Currently, four DON Research and Development (R&D) centers use ALS/N in a test and evaluation mode. The DON MCCR waiver process now

includes ALS/N consideration as part of the standard acquisition formula for both new starts and upgrades.

I.12 AVIONICS PROJECT

The avionics project is a major system upgrade for an airborne Command, Control, and Intelligence (C2I) application that targets existing platform and potential forward fit into next-generation aircraft. The upgrade is to improve acoustic and nonacoustic processing capabilities as well as signal processing, detection and classification, multistation integrated systems, data buses, and communications.

Lesson 1: *Ensure that software production or cost modeling includes adequate time for the Requirements or Design Phase before accepting externally generated completion dates.* The contract was awarded in July 1987 with a prototype scheduled for delivery in July 1990. An optimistic production of 1.2 million SLOC is projected.

Lesson 2: *Be sure that requirements are fully defined and are traceable to test mechanisms. Include necessary Government visibility into the process. Beware of shortcuts and bad engineering practices, especially when there is a prime contractor-subcontractor team organization.* The Firm Fixed Price (FFP) contract included production options. The contract options were tied to calendar exercise dates, without a requirement to demonstrate performance capabilities.

Lesson 3: *Do not plan to use equipment that is under development unless absolutely necessary. Apply a risk engineering approach to those items that must be used, place items on a critical path, and monitor them closely.* The contract included the planned use of "in-development" Government-Furnished Equipment (GFE) and Contractor-Furnished Equipment (CFE).

Lesson 4: *Always assume that everything could go wrong and perform full risk engineering and management.*

Lesson 5: *Use a hands-on management approach from both the prime and Government perspectives and delineate clear lines of authority and responsibility for contractual requirements, especially for large projects.* In addition, do not take a hands-off approach to subcontractor management.

Lesson 6: *Specify in the contract requirements that capabilities must be established early, with adequate resources and authority. Closely monitor progress.* A plan must be developed for handling distributed development environments and deliverables exchanges. Such planning must have been contractually required and completed, and it must receive some degree of Government approval and monitoring before the program is executed. A "sell-off" from a subcontractor to the prime contractor must address all contingencies

when the prime contractor-to-DON delivery requires changes, retesting or documentation, and the like. Configuration management and QA should be standardized and coordinated across the whole effort. Formal, standardized software development procedures should be specified in the contract and approved before being implemented. Lack of such formal, standardized procedures cannot be condoned, especially across larger projects. The procedures should be monitored to ensure that the documented process is being implemented.

Lesson 7: *Do not disregard the critical elements of the MIL-STDs unless it is technically and managerially necessary to use alternative means. Develop a system-wide integration plan and follow it.* During the development of the avionics project plan, a systemwide integration plan was not developed.

Lesson 8: *Ensure that the schedule can accommodate slack and the possibility of independent DON test time for interim products. Also ensure that the resources are available to support regression testing.* The avionics schedule contains no plan for slack or for resources to support regression testing.

Lesson 9: *Do not disregard critical MIL-STD interim products in the contract requirements, and adequately plan for and execute the Government's role to ensure quality and delivery.* Mutually agreed-to criteria for major milestones must be met, or action item work plans must be created for unmet criteria.

Lesson 10: *Ensure that adequate development support facilities exist. Existence of these facilities should be contractually specified and monitored during the Product Readiness Review (PRR). Contingency plans should be available when and if problems develop.* Inadequate facility estimates, combined with no forward-looking projection analysis and unavailability of contingency plans, resulted in severe problems as the interim product grew in size.

Lesson 11: *Do not let events external to the schedule influence the program. Develop input and output criteria for major milestones and adhere to them. It is very easy to build the wrong software.* During the avionics project, time spent in the Requirements or Design Phase was insufficient to mature the software baseline.

Lesson 12: *Where possible, use real production hardware and/or commercial prototypes to decrease the amount and scope of simulation.* The simulator software must be treated as critical-path material if it is to be used during development. Simulator software also should be documented as operational software because it will be critical when mission requirements are being tested. (For example, the system may function in a simulator environment but fail in the real world.)

Lesson 13: *Do not approve systems until requirements are met because when system requirements are not met and "as-built" systems are approved, the contractor is no longer responsible for fixing the system.* The system should not be approved until requirements are met. Design information should not be placed in Software Requirements Specifications (SRSs) and Interface Requirements Specifications (IRSs).

I.13 PEO-SSAS, PMS-414, SEA LANCE

The SEA LANCE Anti-Submarine Warfare Standoff Weapon (ASWSOW) was being developed to provide Vertical Launching System surface combatants and nuclear power attack submarines with a standoff-range missile for use against hostile submarines. Before partial program termination in December 1989, the program was in Full-Scale Development (FSD).

SEA LANCE is a long-range ASW missile system developed to complement ship-launched torpedoes and helicopter-borne weapons by providing a quick-kill opportunity at long ranges. SEA LANCE also can be launched in a buoyant protective capsule that floats to the surface from a submarine torpedo tube. The tactical missile employs seven embedded processors for providing guidance, navigation, and flight control functions. These tactical processors are the Guidance Electronics Unit (GEU), which uses a Motorola 68020/68881 processor; the Inertial Measurement Unit (IMU), which uses a Zilog Z8002 processor; the Pulse Driver Unit (PDU), which uses an INTEL 8797 processor; and four Fin Actuator Units (FAUs), each of which uses an INTEL 8797 processor. Software has been developed under the guidelines of DOD-STD-1679 for each of these subsystems, the most extensive development effort being for the Guidance, Navigation, and Control Program (GNCP) in the GEU.

SEA LANCE system software consists of the embedded GNCP; three embedded small systems software programs (IMU, PDU, FAU); two embedded instrumentation/flight termination system programs; and missile test set, support, simulation, and adaptor/interface electronics software. Ada was used as the PDL and the high-order implementation language only for the development of the GNCP. The following languages were used in all of the other SEA LANCE software development efforts: IMU—Z8000 Assembly; PDU—PL/M 96; FAU—PL/M 96; Arm and Control Unit—PL/M 96; Instrumentation Data Unit—68020 Assembly; missile test set software—Pascal; support software—Pascal, FORTRAN, and Assembly; simulation software—FORTRAN and specialized languages. All discussion and lessons learned are concerned only with the GNCP.

The GNCP is a digital computer program totally contained in nonvolatile memory, which resides in the missile's GEU. It consists of approximately 20,000 SLOC (100,000 physical SLOC). The GNCP was being developed in accordance with the guidelines of DOD-STD-1679 using the VADS. Before program termination, the GNCP had

successfully passed through program milestones such as Preliminary Design Review (PDR) in August 1984, a Delta-PDR in February 1988, an In-Process Review (IPR) in March 1989, and numerous Technical Interchanges between 1983 and 1989. Draft versions of a test specification, test plan, and test procedures were developed in parallel to the design. The GNCP was developed, tested, and integrated at the module and system levels in the contractor's Computer Program Development Laboratory (CPDL), Operational Mock-Up (OMU) Laboratory, and System Integration Laboratory (SIL). Performance and most preflight testing of the GNCP was done in the SIL to fully exercise each function specified by the performance specification. The GNCP guided the test missiles along two near-perfect trajectories in the only two SEA LANCE Contractor Test and Evaluation flight tests in February 1990.

Because the GNCP had not yet reached CDR at the time of program termination, DON never approved or accepted it. As part of the partial termination efforts, the GNCP design of record was documented in accordance with DON direction and archived.

As part of the partial termination efforts, a DON/Boeing study is in process. This study shows the impact of switching to the newer defense software development standards (DOD-STD-2167A and DOD-STD-2168). The study is being conducted in accordance with the guidelines of Military Handbook (MIL-HDBK)-287.

Lesson 1: *Use a consistent methodology throughout the program Requirements, Design, and Coding Phases to facilitate tracing requirements to the code.* SEA LANCE used a functional decomposition method in developing the requirement specifications, then used an Object-Oriented Design (OOD) methodology when developing the design specification and the code. The two methods had to be combined. Because SEA LANCE was a fire-and-forget weapon, the traceability of every performance requirement was considered extremely important. Use of two design methods made it difficult to trace the requirements from the Performance Specification into the Design Specification and then into the code itself.

Lesson 2: *Use a common PDL across the project. On medium- to large-scale systems, the PDL will contain a wide variety of differing coding techniques and code fragments.* SEA LANCE used Ada as its PDL. It was learned that when Ada was used as a PDL, the software development and uniform coding standards should be enforced on the PDL as well as on the actual Ada code.

Lesson 3: *Include and enforce a requirement for a minimum ratio of 50/50 comments-to-code in the contract, software development plan, or coding guide.* Although Ada is more readable than many other languages, it still requires a liberal use of comments to describe what is going on and why. Generally, Government code reviewers needed more review time because of the lack of comments.

Lesson 4: *Use an automated format utility or equivalent software tool to ensure uniform code appearance. This can be imposed through either QA or configuration management.* The SEA LANCE contractor did not always use a printer format utility or other automated tools to ensure uniform appearance of the code. As a result, many Ada specifications and bodies had a unique appearance, depending upon the individual coder.

Lesson 5: *Develop a style guideline for the Ada code and PDL before doing any design work.* The SEA LANCE contractor developed most of the PDL without a formalized Ada coding guideline. The result was a PDL that sometimes differed from module to module in appearance, style, and coding format.

Lesson 6: *Use software metrics from the beginning and define basic terminology between Ada and the selected software development standard.* The minimal use of software metric tools and the defining of basic terms in the early development process gave rise to conflicts between the contractor and the Government as to what constituted a module, a line of code, or the difference between a PDL line of code and an operational line of code.

Lesson 7: *Hammer out documentation requirements and licensing agreements between the Government and the contractors regarding the use of third-party software and the way it is to be tested and identified.* The SEA LANCE contractor employed a proprietary third-party ARTX, and the Government had trouble obtaining documentation on the inner workings and testing of the Run Time Executive software.

Lesson 8: *Early in the development process, have the contractor provide a detailed list of tools that will be used in the development process for the PDL/code and specify the format that will be used for transfer of source code, executable code, and software documentation to the Government. (Note that DOD-STD-1679 did not require a Computer Resource Integrated Software Document [CRISD].)* The Government had some difficulty finding compatible computers to load in contractor-transferred software listings. It also proved difficult to identify the exact format of software deliverables and the exact configurations of the contractor-used development tools.

I.14 NAVY WORLD WIDE MILITARY COMMAND AND CONTROL SYSTEM (WWMCCS) SITE-UNIQUE SOFTWARE (NWSUS) PROJECT MISSION

Lesson 1: *It is always safer to build and test incrementally.* Space and Naval Warfare Systems Command (SPAWAR) PMW 161-5 is responsible for modernizing eight existing site-unique COBOL 1968 applications with approximately 300,000 lines of Ada source code on the NWSUS project. These applications are operational on the WWMCCS Honeywell DPS8 mainframe and are being reengineered using Ada OOD with DOD-STD-2167A because Honeywell is phasing out maintenance of COBOL 1968. This

is within the WWMCCS Automatic Data Processing (ADP) Modernization (WAM) Program. The NWSUS project, which is divided into three increments, is in the third year of a 5-year effort. The first increment consists of six smaller applications with the larger applications in the later increments.

Lesson 2: *Planning for and designing in reuse yield long-term benefits.* The project is in accordance with DOD-STD-2167A/2168 tailored for OOD. The existing COBOL applications are used to capture requirements. Development is performed on a Rational R-1000 model 40 with Honeywell DPS8 and IBM PC/XT clones as targets. With one exception, the applications are Management Information Systems (MISs), and the development makes extensive use of a common set of reuse components.

Lesson 3: *For large software undertakings, use of automated tools is mandatory.* The 2167A documentation is being developed on the Rational, and a Computer-Aided Software Engineering (CASE) tool has been developed to validate the completeness and consistency of the requirements, design, object/class specifications, and Ada specifications. Two "4GL-like" productivity tools, used in conjunction with the reuse components to create application screens and reports, are used for rapid prototyping and to support the generation and standardization of the user interface.

Lesson 4: *Until the design baseline has been approved and frozen, it is inadvisable to initiate full-blown coding.* An initial CDR was completed for Increment 1 in April 1990, and a second CDR to review redesign caused by a change of target was conducted later. Development of many of the reuse components was completed. Full development of the Increment 1 Configuration Items (CIs) began and was completed in FY92.

A full Object-Oriented Requirements Analysis (OORA) and specification for the Increment 2 CIs were completed at the System Design Review (SDR), which was very successful. Both the site customer and SPAWAR commented on the effectiveness of OORA. The CDR occurred in October 1991.

Lesson 5: *If a risk engineering approach (i.e., awareness, identification, technical management of alternative solutions) is taken to development, then it is possible to undertake technologically challenging developments.* Conventional wisdom says that a project with a new application area, a new programming language, or new personnel will have trouble. NWSUS had all three; consequently, the project has had its share of problems. The problems spanned development methodology and standards, target development environment (both Ada compiler problems and problems with the compiler/operating system bindings), Ada training and startup, software reuse, contract structure, and management. However, NWSUS has managed to survive these problems and is currently in a productive mode.

The following lists some of the problems encountered and their solutions or workarounds.

Problem	Resolution
Ada compiler was unavailable for Honeywell DPS8, and WWMCCS Information System (WIS) Workstation target was unavailable at contract start.	The Rational was selected as the host development environment for all applications. Testing is first done on the Rational and then on the target.
Functional analysis was required for the first increment.	The functional analysis approach did not work out well. Full object- oriented analysis was used for the second increment, and that approach has been very beneficial.
The contract assumed that all CIs were the same, and a hard split between design and code hindered Ada OOD.	The contract structure was modified to reflect the diversity of the CIs and the R&D nature of the project and to allow an efficient mechanism for reuse components and prototyping.
Contract and management of reuse between applications initially was weak and/or missing.	An internal approach was used to support reuse on a level-of-effort Work Breakdown Structure (WBS). DON recognized the need in the contract update.
DPS8 Ada compiler was late and not mature; the WIS Workstation was canceled.	The workstation target was changed to a PC. Redesign is under way for the new target and for a solution of the problems encountered with the DPS8 Ada compiler.
Initial training was affected by the "3-week syndrome."	Initial training was too compressed and not project specific. NWSUS now uses a part-time, 2-month, in-house training seminar with a "lab session" that uses project deliverables.

OOD proved to be labor intensive during the first increment.	Ada OOD proved to be a very effective development approach because it gives much more visibility and control of the analysis and design. The drawback is that this requires much more effort. We found no available CASE tools that supported it, and too much had to be done manually. The validation process was automated for the second increment.

I.15 EVENT-DRIVEN LANGUAGE/COBOL-TO-Ada CONVERSION PROGRAM

From 1987 to 1989, the Marine Corps replaced its aging inventory of ruggedized IBM Series-1 minicomputers with hardened IBM-compatible microcomputers. The transition required that all of the systems originally programmed for execution on the Series-1 be ported to the microcomputer. Approximately 25 systems were written in Event-Driven Language (EDL) or COBOL. At about the same time, Ada was introduced as the standard programming language for DOD. The close proximity of the two events provided the Marine Corps with an opportunity to gain valuable expertise in the new DOD standard programming language through reverse engineering of well-known systems. At the time, the Marine Corps had no in-house Ada programmers and no expertise in its associated design methodologies.

The reprogramming effort was divided among three Marine Corps Central Design Programming Activities (CDPAs) along functional boundaries. In the process of the reprogramming effort, the Marine Corps learned several lessons.

Lesson 1: *Training is essential for both technical and management personnel.* To take full advantage of Ada, designers and analysts must be familiar with the principles of software engineering and the way Ada supports those principles. Because few Marines had knowledge of Ada design methodologies at the outset, the tendency was to recode the original system designs in Ada. The original system designs were often derived directly from the existing EDL/COBOL code. Because neither of those languages contains all of the Ada constructs, the advantages of Ada did not always materialize.

Lesson 2: *Programmers require 4 to 9 months of training before they become proficient.* It takes 4 to 9 months of formal and on-the-job training before a programmer becomes proficient in Ada. However, after that initial training period, the programmer should be capable of producing code very rapidly when given a good design and programming library.

Lesson 3: *Military transfers often result in a loss of investment in Ada training.* Because proficiency in Ada can take as much as 9 months to attain, a newly trained programmer is productive only for a portion of his or her tour. Unless steps are taken to ensure reassignment to another Ada shop, the training investment is likely to be lost.

Lesson 4: *Systems originally written in languages that predate Ada that must be converted to Ada should be redesigned, not translated.* After the first few projects, it was evident that inefficiencies in the original designs were being duplicated in the Ada translations.

Lesson 5: *Ada facilitates reuse.* During the conversion effort and on subsequent projects, the Marine Corps found that on an average project, only 45% of the code had to be written from scratch; the other 55% came from reuse. Reusable code generally came from previous projects and development tools (e.g., AdaSAGE). In recent projects, the Marine Corps has consulted Ada software repositories for reusable code in an effort to reduce development time and effort wherever possible.

Lesson 6: *Ada lends itself to efficient code and high programmer productivity.* The syntactical structure of Ada helped the Marine Corps implement many of the software engineering principles. Modularity, information hiding, localization, and abstraction were easily implemented.

Lesson 7: *Development tools are essential.* Initially, lack of a good tool kit hindered the conversion effort. In-house tools were built to overcome Ada file limitations and to enhance screen management. Shortly thereafter, the Marine Corps funded the development of AdaSAGE, which reduced development time by as much as 50%.

Lesson 8: *Development and maintenance time can be significantly reduced by applying software engineering principles and capitalizing on reuse.* The Marine Corps estimates that from 15% to 60% reduction in development and maintenance time are being achieved when software engineering principles and reuse are applied.

I.16 SHIPBOARD GRIDLOCK SYSTEM WITH AUTO-CORRELATION

The Shipboard Gridlock System with Auto-Correlation (SGS/AC) application plays a fundamental role in the coordination of multiplatform shipboard systems by processing the ships' data and remote track data within a common positional frame of reference. This application performs gridlock processing to correct for sensor and navigational errors while correlating the identified tracks from remote systems. This software-based application is characterized by hard deadlines; multiple external interfaces; and time-critical, computationally intensive processing. The SGS/AC is deployed on the Aegis cruiser/destroyer class of surface ships.

Lesson 1: *Before commitment is made to large projects, the methods and tools to be used should be exercised. Quantitative evaluation of the expended resources should lead to better estimates for the work contemplated.* This project is being performed by the Naval Surface Warfare Center (NAVSWC). It can be characterized as a DEMVAL development effort that parallels the SGS/AC program implemented in CMS-2 for either the AN/UYK-20(V) or the AN/UYK-44(V) target processors. This parallel effort uses ALS/N as the host development tool set and targets an AN/UYK-44(V) processor configuration. An additional objective of the effort is to generate a comprehensive comparative analysis of the CMS-2 and Ada developments that includes quantitative data and information pertinent to future Aegis-class combat direction system upgrades.

Lesson 2: *The Ada code itself will have major architectural and design impact on a system; therefore, the two must be worked on simultaneously.* From the outset, it was recognized that to simply translate CMS-2 code to Ada would be technically feasible but would not produce any long-term benefit.

Lesson 3: *A project should always try to build a little and test a little, building and testing the harder things first (e.g., system services and communications).* The new design effort attempted to minimize the run-time overhead, include portability in the design, manage interfaces to get best-case response under worst-case loads, and maximize robustness and predictability. A multiphased build plan was initiated.

Lesson 4: *A project should always attempt to involve the production hardware as early in the program as feasible. Successful simulator and emulator runs mean nothing when the delivered code does not work on the real hardware. Acceptance requirements must be set correctly, or development schedule reserve must be allocated to absorb such difficulty. Things will go wrong, and this should be anticipated.* / development is being carried out on VAXs, with DEC Ada being used during the ea Code and Test Phases. The target AN/UYK-44(V) processor requires special cards to run the Ada code. The particular configuration was unavailable until well into the project.

Lesson 5: *The team must be well trained in the use of the supplied tools, and the tools must work as advertised.* The ability to fully define a working set of integrated tools early in development and to acquire them as they are needed is critical. For example, a symbolic debugger is an absolute necessity.

Lesson 6: *Adherence to good engineering practices is necessary when designing the system and its hardware and software.* Although this project is a relatively small software undertaking, establishing and enforcing sound software design methodology and development processes, such as coding standards, documentation production, and code reviews, help overcome lapses in memory, personnel turnover, lack of focus, and lack of requirements to trace verified design or code.

Lesson 7: ***Until more technological progress is achieved, the potential for large-scale software component reuse is limited.*** This project has shown that achieving real-time developments requires meeting hard deadlines and getting close to the target machine, which often conflicts with the concept of code component reuse.

I.17 SUBMARINE COMBAT CONTROL SYSTEM MK2

The Submarine Combat Control System (SCCS) Program focuses on consolidating the various Combat Control and Defensive Weapon Systems (DWSs) software configurations that are in use on deployed SSN-688 and SSN-726 class submarines. These vessels constitute both the defensive (attack) and strategic platforms for the DON submarine force. The SCCS upgrade will either upgrade or replace obsolete general-purpose computers, peripherals, display consoles, and weapons simulators. This software upgrade provides a common software package for both classes of submarine and incorporates operational and maintenance-related enhancements. The SCCS Program also includes the development of systems to support crew training and land-based testing.

The software for the SCCS consists of new development software and firmware, modified Government-Furnished Software (GFS) and firmware, and unmodified commercial software and firmware.

Most of the modified GFS software has been written in either DON-standard CMS-2 HOL or in ULTRA-32 Assembly. The project mission is to develop a maintenance capability that improves the chances for coordinating evolutionary change in these shipboard systems.

The new portion of the SCCS MK2 program involves integrating a replacement human-computer interface display console and associated Ada application software into the existing deployed systems. The approximate language mix is as follows:

Language	SLOC
CMS-2 & ULTRA-32	2M (GFS/modified)
Ada	581K (new)
C	279K (commercial)
FORTRAN	149K (retained)

The Ada SLOC are being developed under DOD-STD-2167A requirements. The CMS-2, FORTRAN, and ULTRA-32 software were all developed under DOD-STD-1679A.

The paragraphs below summarize the lessons learned about Ada on this project.

Lesson 1: *Ada experience and training are needed.* The majority of experienced personnel in this defense area had little or no experience with Ada and modern software engineering practices. It was necessary to evaluate bidders on their in-place Ada expertise and on their ability and/or plans to acquire or build on that base. To properly monitor or manage the development, in-house capabilities had to be built up in these areas. It is especially important to use hands-on training as close to development as possible or during development.

The relative immaturity of candidate Ada products, coupled with the specific need to handle many foreign language interfacing requirements, meant that the developer team needed a very close relationship with their candidate Ada development tool suppliers.

Lesson 2: *Support software, practices, and products need constant attention.* This undertaking required that the chosen contractor be capable of using automated tools to manage and technically execute this large programming development. To that end, source selection criteria were established and used during the source selection process.

Each project has to generate its own Computer Resources Life-Cycle Management Plan (CRLCMP) and Integrated Logistics Support Plan (ILSP) before the Defense Acquisition Board (DAB) Milestone I. However, unless the Government defines the total development environment fully and requires its use as part of the proposal, difficulty will ensue as differences develop between the methodology, tools, and equipment used by the developer and those specified by the Program Office. Typically, the parties involved will have opposing agendas. Coupled with the inability of many tools to scale up to programming-in-the-large or even to exchange data structures efficiently, this diversity will create problems that all parties will need to address and work out on a continuing basis. Examples of areas where this problem resolution may be required include tool standardization; data exchange; version management; electronic communication; data rights; documentation uniformity; configuration management; error identification, analysis, and elimination; product ownership; component integration; and testing.

Lesson 3: *The need to interface with other language programs may constrain the type of Ada features that can be used.* The Ada language design run-time concept does not map directly to the hard real-time environment within the MK2 system. Therefore, attempts must be made to overlay the Ada model on top of the inherited real-time operating system, which has necessitated eliminating certain Ada features (e.g., tasking). Other Ada features not used include generics, dynamic allocation, and full-range data typing. Performance also has suffered, and portability has been minimized. The need to interface with other language programs may result in a loss of the advantages of strong Ada typing and may affect debugging, testing, certification, and the like.

Lesson 4: *To ensure programming uniformity, a style guide should be developed and used across all developer teams.* Use of a common style guide will enhance overall maintainability of developed code. It also will help control Ada feature utilization, and the code can be automatically checked by applying a preprocess tool. The use of a "pretty printer" postprocessing mechanism for human-readable outputs could also enhance software maintainability.

I.18 P-3C UPDATE IV Ada DEVELOPMENT

The objective of the P-3C UPDATE IV Program is to develop a fully integrated, distributive bus, data processing system with improved mission avionics systems. The full weapon system is to be tailored for both retrofit into P-3C predecessor aircraft and forward fit into successor Maritime Patrol Aircraft. The program successfully progressed through the DEMVAL Phase between November 1984 and April 1987. After the Milestone II decision in July 1987, Boeing was awarded an FFP contract for FSD to develop and fabricate the system, qualify vendors, install the system into a P-3C platform, and conduct vendor flight tests by July 1990. The schedule called for Government testing of the flying test bed between July 1990 and February 1992 with subsequent approval for full production to be granted in April 1992.

The program includes the distributive bus data processing Distributed Processor/Display Generator Unit (DP/DGU) system, which consists of six Motorola 68020-based processor modules/DGUs tied together by a dual 1553B bus architecture. Major mission systems avionics include the AN/UYS-2 acoustic processor, the Motorola 68020-based AN/ALR-66(V) 5 Electronic Support Measures (ESM) system, and the AN/APS-137 (V) 3 Inverse Synthetic Aperture Radar. The data processing system and ESM are CFE, and the acoustic processor and the radar are GFE.

The program has been delayed by both hardware and software development difficulties. Boeing was expected to deliver the flying test bed to the Government between October 1992 and February 1993.

As one of the first large Ada developments (over 1 million SLOC), the P-3C Update IV program has been a pioneer in the use of Ada. Boeing personnel have made several correct choices in developing software in a new programming language for which the software development environment was immature or limited. First, Boeing's choice of using the VADS was a good one. VERDIX has been a leader in the development of Ada software engineering tools, and VADS was one of the best Ada software development environments available at the time of program initiation. Equally good was the choice of the Ready Systems kernel as the core for the operating system. Finally, Boeing's naming convention for Top Level and Lower Level Computer Software Components (TLCSCs/LLCSCs), packages, units, and identifiers has also been beneficial. The naming convention has been very useful in tracing requirements to design and code and

is helpful when reading the PDL and computer source code. The paragraphs below summarize the lessons learned about Ada use in this program.

Lesson 1: *Ada code requires more up-front time and effort, and the learning curve is slower.* The software size and development schedule estimates were understated by all parties during the initial phase of the program. The table below lists SLOC estimates at program initiation, at completion of PDR, and in August 1991.

The final SLOC total should exceed August 1990 estimates by more than 10% before completion of software development. The initial sizing estimates will be in error by approximately 100% at program completion.

The Boeing estimates for the software development schedule were predicated on available non-Ada HOL usage. Individual task estimates were too short and did not anticipate the increased up-front work in Ada design and coding that was needed. This fact and a slower than anticipated learning curve for coders resulted in a realized progress rate of 85% of plan for coding, testing, and integration activities.

Lesson 2: *Increased facilities and memory are required to accommodate Ada code.* The physical number of hardware tools was initially insufficient to support a software development of this magnitude. This lack of hardware capacity was experienced in all areas of the software development environment, from the Sun workstations used during initial code and testing to the System Avionics Integration Laboratory (SAIL) used for system integration. More Sun workstations were needed to avoid bottlenecks in coding, both in the SDL and at the subvendor locations involved in tactics and correlation programming efforts. The Boeing SDL grew from two Sun 3/280 server stations with 33 Sun client workstations in the fall of 1987 to five Sun 3/280 server stations with 45 Sun client workstations in the fall of 1990.

The SDL mainframes used for the target hardware software build process could not construct a software build in an acceptable period. Initial software program builds took up to 1 week to compile and link. The SDL initially contained one VAX 11/785, one VAX 11/750, and two VAX 8700s. To accommodate the software development demands, the SDL was upgraded by the fall of 1990 to include one VAX 11/785, one VAX 11/750, two VAX 6000/440s, and one VAX 8600. Disk storage capacity was also increased to approximately 40 gigabytes (GB). This increase in hardware capacity has reduced system build time to approximately 8 hours.

Initial plans called for target integration to be conducted on a single SAIL that contained as much actual UPDATE IV hardware as possible, including the full DP/DGU system. A DON SAIL was held at Boeing instead of being delivered to DON to accommodate the effect of the integration overload on the Boeing SAIL.

Computer Software Configuration Item (CSCI)	Best and Final Offer (BAFO) (April 1987)	June 1988	August 1991
DP/DGU	383,530	468,654	565,431
Minimum Mode Software (MMS)	0	37,900	52,764
Electronic Support Measure (ESM)	52,340	58,000	55,516
Acoustic Interface Unit (AIU)	68,900	68,900	97,371
AN/UYS-2	37,320	38,000	147,500
System Avionics Integration Laboratory (SAIL)	218,600	172,870	211,766
Integration Test Software (ITS)	5,000	96,900	109,359
Software Development Laboratory (SDL) (Boeing-Developed Code Only)	37,500	45,500	62,800
TOTAL	803,190	986,724	1,302,507

Lesson 3: ***Some software development tools are immature and have not been proven for many applications.*** Immaturity and/or unavailability of software development tools also complicated early software development efforts. A comprehensive Ada support environment was unavailable for early development work. Available tools were immature and were not integrated into a comprehensive package. In addition, available tools were very resource intensive, which exacerbated the previously mentioned hardware problems.

The Sun workstation software build installations initially required 1 week and contained numerous errors because of excessive operator intervention. Upgraded Sun workstation software and software tool/automation development resulted in eventual turnaround times of 1 day. Error reduction was excellent as a result of the automated tools.

The initial SDL VAX systems were plagued with software and hardware faults, which resulted in numerous system crashes and an average downtime of 1/2 day per week. By applying pressure to Digital Equipment Corporation, fixes were put in place over a period of 1 year, which resulted in mature, stable system performance.

The VERDIX compiler was selected for use after available compilers were screened by the procedures recommended in 1987. However, numerous early software and hardware errors were encountered before stable performance was achieved. As late as January 1991, the VERDIX Ada compiler with the version 6.0 program was found to have an optimizer error. After the compiler was corrected, the UPDATE IV program required a total recompile to remove inefficiencies scattered throughout.

Lesson 4: *Tailoring of DOD software development standards must be addressed to accommodate Ada-unique capabilities.* Although DOD-STD-2167A does not require that software development efforts follow the traditional waterfall model associated with DOD software developments, it does not provide guidance on alternatives. Ada forces more detailed design earlier in the software developments than do previous languages because of the Ada package specifications and the strong data types imposed by Ada. These factors encourage a pseudo "rapid prototyping" approach rather than the traditional waterfall during the design phases.

DOD-STD-2167A does not address distributed processor systems or multiple CI developments. Ada was designed specifically for a modular approach to large software developments. For example, DOD-STD-2167A does not adequately address testing among multiple CIs or the Integration Phase issues. DOD-STD-2167A documentation neither reflects Ada terminology or structure nor addresses an appropriate approach to documentation development.

Lesson 5: *Although the adoption of Ada was envisioned to enhance the software development process, use of Ada does not guarantee sound software engineering practice.* Specific areas where Ada does not substitute for sound engineering practices include:

- *Establishment of system and software requirements.* A Requirements Analysis Phase must be conducted to produce appropriate system-level requirements that are then allocated to hardware and/or software as appropriate. Participation by both contractor and Government systems engineering personnel throughout this evolution is critical to program success.

- *Enforcement of control points.* The contract must require and the Government must enforce a variety of control points. These control points must take into account Ada-unique development approaches where the approach differs from the traditional DOD-STD-2167A waterfall model. Allowing the contractor to proceed past these control points, even if he does so "at risk," imposes significant risk on successful program completion.

- *Configuration management.* Use of Ada does not preclude Government requirements for establishing and controlling the functional, allocated, and product baselines. Use of Ada may complicate control of the software allocated baseline by inviting inclusion of design detail into the software requirements documents. Although Ada forces more detailed design earlier in the software development process, the temptation to include this detail into the software allocated baseline must be avoided.

- *Testing.* The mapping of Ada constructs to DOD-STD-2167A "units," "modules," and "system" is imprecise and can lead to inadequate testing of Ada code. The DOD-STD-2167A premise of fully qualifying a software entity at one level of abstraction before combining that entity into larger integrated components should be maintained. A software entity should not be considered fully qualified solely because the higher-level entity into which it is incorporated successfully passes its qualification requirements.

Lesson 6: ***Strict configuration management and control are required to enforce discipline to counter complexity-induced confusion.*** Lack of familiarity with Ada, a slow learning curve for new coders, and schedule delays reemphasize the absolute requirement to maintain strict software and hardware control within all facilities. With differing levels of coding, unit and package testing, informal integration testing, and formal systems testing occurring in the respective facilities, strict configuration management within the facilities and within the software development library was mandated. The Government audited the initial Software Development Folders (SDFs) and found them deficient. Replication of numerous informal tests could not be accomplished from the SDFs, as written.

I.19 STANDARD FINANCIAL SYSTEM REDESIGN

The Standard Financial System (STANFINS) is part of the total U.S. Army accounting system and serves as a field-level system for general funds servicing posts, camps, and stations. The original STANFINS was a batch processing system written in COBOL. A STANFINS Redesign project (STANFINS-R) was undertaken to overhaul the system and make it interactive.

STANFINS-R consists of two subsystems—Subsystems I and II—to be developed independently. This large system is designed to handle mainstream accounting applications such as the general ledger, accounts receivable, fixed assets, and cost accounting standards. The system consists of 500 programs with approximately 2 million lines of code, and it generates 147 reports. The contract for developing Subsystem II, which originally was viewed as a large COBOL project, was awarded to the Computer Sciences Corporation (CSC) in the fall of 1986. However, the contract was modified in the spring of 1988, and Ada was designated as the development language. The first pilot teams were formed in the spring of 1988, and the actual writing of code and Ada bindings began in the fall of 1988. Software development testing and software qualification testing started in the summer and fall of 1989, respectively. Most of the system tests have been completed, and part of the project is operational.

The project was developed in an automated program support environment composed of six Rational R-1000 machines. The code was eventually ported to the target environment, an IBM mainframe running OS/VMS.

Despite delays in the implementation schedule and budget overruns, the STANFINS-R project indicates that there are several advantages to using Ada in information systems development. For example, programmer productivity has been quite high (594 lines of code per staff month), almost double that of typical COBOL projects, and the quality of the software, as evidenced by the test results, appears to be uniformly high.

In many ways, STANFINS-R is a prototypical information systems project from which many lessons, including those described below, can be learned about Ada use.

Lesson 1: *The available pool of developers skilled in Ada is limited.* When making projections about project costs, the issue of the limited number of available personnel skilled in Ada and the need for training should be considered. STANFINS-R originally was conceived as a COBOL project. When denial of the waiver resulted in a switch to Ada as the development language, it became apparent that the available pool of developers with Ada/MIS experience was small. The existing staff of COBOL programmers had to be trained in Ada, which caused delay in project execution.

Lesson 2: *Few compilers and support tools are available for information systems development in the IBM environments that use Ada.* STANFINS-R demonstrated that Ada is a viable language for developing information systems in environments where COBOL has been the dominant development language. However, the IBM environment, which is the primary environment for developing such systems, is poorly supported in terms of compilers and support tools. STANFINS-R was the first Ada application of its kind and size to be developed to run on an IBM OS/VMS environment. Because of the lack of available tools to support Ada in this environment, a set of support tools, such

as code generators and screen painters, had to be developed as part of the project. Moreover, the compiler, which was developed by Intermetrics but had not been validated, did not provide support for a CI-based teleprocessing monitor; therefore, the contractor had to write one. In addition, the DBMS package chosen for the project (i.e., Datacom DB) did not contain a suitable Ada interface; therefore, a hook had to be written. For Ada to be a feasible language for use in developing information systems, the issue of availability of compilers and support tools must be addressed. Most Ada vendors do not offer products in this environment. The dominant compiler in this environment (offered by Intermetrics) has not been validated. The unavailability of suitable compilers has been a significant factor inhibiting the use of Ada in information systems and has created an adverse cycle of events. On the one hand, because Ada is not the preferred language in information system development, vendors have little incentive to offer products to work on the platforms on which such applications are traditionally developed. On the other hand, the paucity of suitable products works to limit the consideration of Ada in developing information systems. Successful implementation of the mandate to use Ada will require a suitable resolution of this cycle. A plausible way to address this problem would be to create appropriate incentive structures that will encourage vendors to develop such products.

Lesson 3: ***Systems developed in Ada may be more maintainable than those written in COBOL.*** Although it is too early to state definitively that Ada maintenance requirements are lower than those for COBOL, preliminary evidence indicates this may be so. Part of STANFINS is operational and has a maintenance staff of six programmers, a much smaller team than would be required to maintain a system of similar size that uses COBOL.

Lesson 4: ***In principle, portability is ensured by developing code in Ada; however, in practice, portability is limited.*** Porting the code from the Rational environment to the target environment was problematic. For a variety of reasons, parts of the code that worked well on the Rational environment did not work in the target environment. For example, nested generics would not work on the target although they tested and compiled on the development machine. Executable code sharing could not be implemented on the target, thereby causing the executable sizes to grow to unmanageable proportions. Other features, such as representation specifications, Unchecked_Conversion, and Pragma Inline, were not implemented in the target compiler.

Developers in this project had significant problems with porting code from the development environment to the target environment. What compiled on the Rational R-1000 also compiled on the IBM mainframe using OS/MVS. However, lack of compiler support for the teleprocessing monitor and interfaces to the DBMS necessitated the creation of low-level functionally limited code, thereby limiting portability to other

environments without significant modifications. Thus, while most of the code can be ported to a VAX/VMS environment, for example, complete portability would require significant alterations.

Lesson 5: *Ada has special advantages that make reuse more feasible and enables the benefits of reuse to be realized at several levels.* At one level are specific packages and templates that can be used in other parts of the project or in other projects. While the same could, in principle, be accomplished with code written in COBOL, the use of generics and packages gives Ada a special advantage over COBOL that makes such reuse much more feasible. There was significant use of these templates at STANFINS. The issue of reuse can also be viewed in terms of tools that are developed for specific projects but with suitable modifications can be used in other projects. The STANFINS project, for example, entailed the development of Program Structure Language (PSL)/Program Structure Analysis (PSA) tools for writing design specifications. These tools can plausibly be modified and reused in other projects. A follow-up project, the Standard Army Financial Accounting and Reporting System (STARFIARS), demonstrates reusability at both levels. Although STARFIARS is likely to be at least 33% larger than STANFINS, the project is scheduled to take 50% less time than STANFINS. The rationale for this fast-paced schedule is twofold: STANFINS provided a useful learning curve from which STARFIARS will benefit, and more important, the implementation of STANFINS has created system templates and tools that can be reused to create the new system with greater productivity.

Lesson 6: *Tailoring of DOD software development standards must be addressed to accommodate Ada-unique capabilities.* The documentation required for STANFINS-R, which was prepared according to the requirements mandated in AIS DOD-STD-7935-A, was inordinately large. While the exact figure is difficult to ascertain, a conservative estimate is that every line of code generated at least 10 lines of documentation. The voluminous documentation clearly limits its usefulness and points to the need to reexamine current documentation standards.

I.20 RECONFIGURABLE MISSION COMPUTER PROJECT

The Reconfigurable Mission Computer (RMC) Project sought to demonstrate that modularity in both hardware and software would reduce the cost of developing new or upgrading existing embedded systems. The thrust of the project was to exploit hardware and software commonality in different embedded systems.

Lesson 1: *For small technology demonstration projects, anticipate a lack of Ada compilers for small, embedded computers that use advanced microprocessors.* Primary constraints on missile general-purpose data processors are size, power, and cycles per second. There is always a drive to use the most advanced microprocessors available to get as much performance as possible in as small a space as possible while consuming the

least power possible. Ada compiler vendors, however, are not going to market a compiler until they can determine that it is financially realistic to do so. Small technology demonstrations that want to use Ada in the software development may be restricted to using processors for which a commercial Ada compiler exists.

Lesson 2: *Plan on allocating a portion of the CPU utilization to the inefficiencies of using a modular design approach and design implementation in Ada.* The RMC project goals included creating portable Ada programs, running them on several platforms, measuring the code change required, and learning what it took to make an Ada program portable. A modular design based on Abstract Data Types was used to hide machine interfaces. We also hid the "goodies" the compiler vendors offered outside of the Ada language behind our own package interfaces. The results were a reduction in performance that can be made up with a higher throughput CPU. However, any throughput increase realized by upgrading platforms is usually given to the analyst to develop more capable algorithms. A modular design in Ada can reduce code, test, and modification times and is well worth the extra overhead incurred.

Lesson 3: *Plan on throwing away some or all of the first software designed.* After the first design and implementation of demonstration software in Ada, it was felt that the implementation would be improved the next time. Fortunately, we had the luxury of doing just that, and we understood and implemented a much better software system the second time. It is not necessary to wait until all of the tools and hardware are in place to begin coding. As much of the design as possible should be implemented as soon as possible. A commercial prototype or similar system should be used to gain understanding of the system, and the first cut should be used to verify that the requirements can be met.

Lesson 4: *Dedicate an individual or group (depending on the size of the project) to the Ada-hardware interface.* Ada touches the "iron" in several places: the target debug monitor for on-target program development; the kernel for time, memory, and processor management; and device drivers used by the application. A person or group needs to be familiar with hardware registers, ports, memory locations, and the low-level facilities available in Ada. Evolving hardware architectures and compiler upgrades make this an absolute necessity.

I.21 INTELLIGENT MISSILE PROJECT

The purpose of this project, which is funded by the Office of Naval Technology under the Missile Support Technology block NW2A, is to develop generic software techniques and to design tools that will allow the use of knowledge-based Artificial Intelligence (AI) paradigms for control and decision-making functions in missiles. These capabilities are to be implemented in Ada. Having these features will yield more adaptive and autonomous missile operation.

A simple, forward-chaining inference engine was developed and tested on several computers. Next, a decision-tree type of expert system was developed along with a tool (in Ada) to generate the Ada decision tree. (None of the commercial expert system shells could do this at the time.) Finally, a hybrid system was developed that combined the flexibility of an inference engine with the speed of a decision tree. Execution performance was measured for all three types of systems. The decision tree was the fastest, and the hybrid system was a close second.

Lesson 1: *System analysis must be conducted at the beginning to ensure that adequate resources (e.g., compilers, hardware platform) will be available for meeting the system requirements. The tendency to favor particular hardware or compiler systems just because they are available must be avoided.* Choosing a particular CPU simply because it is available can lead to problems that could have been avoided by performing adequate system analysis. One problem encountered was that the CPU needed an Assembly program to be downloaded and run to "kickstart" the CPU so that Ada code could be downloaded and executed. The CPU was hardwired to have a certain memory configuration that was incompatible with the mandatory location of the Ada code.

Similarly, using a compiler that already is on hand without ensuring it can do the job also will lead to delays. In this case, the compiler had been validated for a particular single-board computer. Although the vendor stated that it should work with the chosen target board, the vendor would not provide any help because compilers for other CPUs had a much higher priority. This happened in spite of the fact that the highest level of maintenance available had been purchased for this project.

Lesson 2: *Although Ada has many features, it does not have every feature from every language; for example, Ada is restricted in the type of AI systems for which it can be used.* Because it is a procedural language, Ada has some restrictions, particularly with regard to certain AI applications. In LISP, an arbitrary string of characters can be handled in three different ways: as text, as a variable, or as a function to be called. This ability, which is very useful for building production-type expert systems, results from LISP's being not only a language but also an environment. Because Ada is only a language, there are restrictions on the types of production expert systems that can be implemented. Although it would be possible to implement the equivalent LISP environment in Ada, LISP is too slow and big for a missile system, which was the reason for its not being used in the first place.

Appendix J

FY91 Ada Technology Insertion Program Projects

This appendix provides a brief description of the Ada Technology Insertion Program (ATIP) projects funded in FY91. The projects fall into three primary categories—education, bindings, and technology. For more information on these projects, contact the Ada Joint Program Office at (703) 614-0209.

J.1 EDUCATION

Of the 14 projects funded, one addresses Ada education. It is the "Undergraduate Curriculum and Course Development in Software" given by the Advanced Research Projects Agency (ARPA).

This program will support the development of educational materials using Ada that will be widely distributed to and used by educators. It will enhance the software engineering content of courses and course sequences in computer science curricula and will demonstrate, through pilot implementations, the feasibility and viability of a comprehensive undergraduate curriculum in software engineering using Ada.

J.2 BINDINGS

The eight bindings projects are grouped into the following categories:

- Government Open Systems Interconnection Profile (GOSIP)
- Management Information System (MIS) Mathematical Binding
- Military Standard (MIL-STD)-1553
- Portable Operating System Interface for UNIX (POSIX)
- Structured Query Language (SQL)
- XWindows.

Ada Application Program Interface to GOSIP Network Services
Defense Information Systems Agency (DISA) (Formerly DCA)

GOSIP is a family of protocols that supports network services. Although Ada bindings to GOSIP exist, this project will develop a robust Ada/GOSIP binding for standardizing the interface of Ada applications to GOSIP network services.

Decimal Arithmetic
U.S. Air Force

Compiler vendors support decimal arithmetic but in nonstandard ways. This project will standardize a mechanism for realizing COBOL-style exact decimal arithmetic in

Ada 83. It will provide sufficient functionality to handle financial applications with at least 18 digits of precision. It will offer early availability with Ada 83 compilers, notational convenience, ease of transition to Ada 9X, and run-time efficiency.

Generic Avionics Data Bus Toolkit
U.S. Navy

This project will offer a standard software interface that can be reused for various MIL-STD multiplex data buses with minimal changes. The initial software will focus on the MIL-STD-1553B protocol because this protocol is the most prevalent, but it will be designed to be configured for expansion to other types of data buses. An integrated MIL-STD-1553B monitor with debugging tools is planned.

POSIX/Ada Real-Time Bindings
U.S. Air Force/Navy

POSIX defines a collection of system services that provide portable application interfaces to operating systems. The POSIX effort is divided into several areas that cover the range of operating system services. These include basic system services, real-time services, security services, user command interface, user graphical interface, network services, mail services, and system administration. This project will develop draft Ada bindings for the real-time service area (POSIX 1003.4 and 1003.4a standards), work with the Institute of Electrical and Electronics Engineers (IEEE) standards organization to promote the use of these drafts as a starting point for development of standard Ada bindings, and develop a test prototype implementation of Ada tasking using the 1003.4 (real-time) and 1003.4a (threads) services.

Ada SQL Interface Standardization
Defense Advanced Research Projects Agency (DARPA)

SQL is a set of standards associated with relational databases and data dictionaries. The SQL Ada Module Description Language (SAMeDL) provides an interface technology for Ada applications accessing SQL database management systems (DBMSs). The ATIP will fully document both the SAMeDL as a language and its supporting methodology, respond to the needs of the standardization process, and coordinate efforts of potential vendors of SAMeDL processors as well as identify needs of potential SAMeDL customers to assist the transition to the SAMeDL.

A SAMeDL Pilot Project on SIDPERS-3
U.S. Air Force/Army

A SAMeDL tool set will be developed consisting of a SAMeDL Module Manager and a SAMeDL compiler. These tools will target a designated database running on an Everex Personal Computer (PC) under UNIX. Both an existing application and a new application will be developed using this tool set. This effort is designed to prove that the SAMeDL tool set has the robustness, maturity, and potential for reusability to be employed as the Ada/SQL binding of choice on any large Department of Defense (DOD), Ada Management Information System (MIS) program.

Common Ada XWindow Interface
U.S. Navy

XWindows is a *de facto* industry standard that provides a Graphical User Interface (GUI). Popular toolkit extensions to XWindows include Open Look (used by AT&T, Sun, and others) and Open Software Foundation (OSF) Motif (used by IBM, Digital Equipment Corporation, Hewlett-Packard, Apollo, and others). This project will design and produce a Common Ada XWindow Interface (CAXI) to both the Open Look and Motif toolkits. The interface will be written in Ada and will allow application programs to use either toolkit without modification to the application program. This will increase the portability of Ada applications and provide flexibility in the selection of hardware.

An Interactive Ada/XWindows User Interface Generator
U.S. Army

This project proposes to develop a general-purpose Ada/XWindows User Interface Generator that automatically generates Ada source code. Using this tool, a developer will be able to interactively develop a functioning user interface by selecting user interface primitives and arranging them on the screen. This tool is intended to reduce the bottleneck imposed upon Ada systems developers when developing window-based user interfaces based on the XWindows system and the Motif toolkit.

J.3 TECHNOLOGY

The five projects in the technology category deal with the following:

- Engineering environments
- Prototyping

- Reuse
- Security.

AdaSAGE Enhancements
U.S. Air Force/Army/Navy

AdaSAGE is an applications development set of utilities designed to facilitate rapid and professional construction of systems in Ada. The Department of Energy developed AdaSAGE at the Idaho National Engineering Laboratory. Applications may vary from small to large multiprogram systems using special capabilities. These capabilities include database storage and retrieval (SQL compliant), graphics, communications, formatted windows, on-line help, sorting, and editing. AdaSAGE operates on various systems including MS-DOS platforms, UNIX System V, and OS/2. A developer using the Ada language and the AdaSAGE development system can design a product tailored to a specific requirement that offers outstanding performance and flexibility. The ATIP proposal provides enhancements to AdaSAGE requested by the user community and supports the creation of a computer-aided training program.

ATLAS/Ada-Based Enhancements for Test (ABET)
U.S. Air Force

ABET is an Air Force and IEEE effort to provide an international standard for an automatic test environment for maintenance activities. Ada is the language to be used for implementing this standard. ABET will intelligently incorporate Ada into the test arena by providing a set of layered standards to the test community.

A Computer-Aided Prototyping System for Real-Time Software
U.S. Air Force

The program will demonstrate a high-technology, low-cost approach to providing the latest software prototyping tools for real-time Ada programs. It provides the opportunity to use the thesis efforts of students at the Naval Postgraduate School, who are DOD personnel familiar with Ada and its embedded applications.

Reusable Ada Products for Information Systems Development (RAPID)
U.S. Army

RAPID is an Ada reuse program that includes an automated library tool for configuration, identification, and retrieval of reusable Ada software components and a staff that supports and trains developers in reusability and sound software engineering principles. Its mission is to ensure that the DOD objective of reusable,

maintainable, and reliable Ada software is achieved. It provides a total reuse program supporting the entire software development life.

Ada Reuse in a Trusted Message Processing System for Real-Time Software
U.S. Navy

This project will investigate Ada reuse in developing software that satisfies the Orange Book B2-Level security requirement. The system will be fielded as the Submarine Message Buffer (SMB) System, supporting personnel with two levels of security clearance.

Appendix K

Navy and Marine Corps Ada Projects

A database of Navy and Marine Corps projects that use Ada has been assembled for reference by Program Managers who are planning to use or currently are using Ada. The database includes the following information:

- Project Name
- Project Description
- Application Area, (i.e., Command and Control [C2], Command, Control and Communications [C3], Command, Control, Communications, Computers, and Intelligence [C4I], Electronic Warfare [EW], Space, Communication, Armament, Ordnance, Acoustic, Navigation, Financial, Personnel, Contracting, Material Management, Medical, Depot Maintenance, Tool, Database Management System [DBMS], Graphical, Education, Simulation, Other)
- Sponsor/Developer
- Point of Contact and Phone Number
- Program Status (in planning, developed, completed, or canceled)
- Source Lines of Code (SLOC)
- Host System
- Target System.

Because this database is very large, its contents have not been included in this version of the *Ada Implementation Guide*. It is available either on disk as a Lotus 1-2-3 file or in hard copy. To obtain a copy, please fill out the attached order form.

If you would like your project to be considered for inclusion in this database, please provide the information listed on the order form.

ORDER FORM

Program Name _____
Prog. Manager _____
Address _____
City, St & Zip _____

Please send:

_____ (1) Copy of DON Ada Projects Database on disk (Lotus 1-2-3 File) and/or

_____ (1) Hard copy of DON Ada Projects Database

. . . . . .

To have your project considered for inclusion in this database, please provide the following information:

- Project Name
- Project Description (brief & concise)
- Application Area, (i.e., C2, C3, C4I, EW, Space, Communication, Armament, Ordnance, Acoustic, Navigation, Financial, Personnel, Contracting, Material Management, Medical, Depot Maintenance, Tool, DBMS, Graphical, Education, Simulation, Other)
- Sponsor/Developer
- Point of Contact and Phone Number
- Program Status (in planning, developed, completed, or canceled)
- Source Lines of Code (SLOC)
- Host System
- Target System.

Please send this order form and/or project information to:

Space & Naval Warfare Systems Command
SPAWAR 3241 (CDR M. Romeo)
2451 Crystal Drive (CPK-5, 700)
Washington, DC 20363-5208

Appendix L

Ada Language Features That Support Software Engineering

Ada has several features that directly support software engineering. This appendix discusses in great technical detail those features that are considered important, including Ada packages, strong typing, exceptions, generics, Ada library (separate compilation), and tasking.

L.1 Ada PACKAGE

Many people consider the Ada package to be the most important feature in the language to support the goals and principles of software engineering. Hence, it is a primary factor in producing software that is reliable, of high quality, within budget, and on schedule.

The Ada package consists of two parts: a ***package specification*** and a ***package body***. The package specification identifies "what" the package is going to do; the package body contains the "how" and provides implementation details of the code hidden from the application. The package specification identifies how any Ada application can interface with the package. In a sense, the package specification is a legal contract with Ada applications. The package body contains the code to conduct the real work of the package. As Figure L-1 depicts, an Ada application must go through the package specification in order to benefit from the code in the body. The specification identifies the only way that an Ada application can interface with the package body.

Typically, an Ada application is a main program with a collection of packages. Attachment 1 to this appendix contains a sample of an Ada package specification. This sample provides an abstraction of the parcel.

Once appropriate abstractions are created with the package feature, the abstractions can be used by the main Ada program or other packages. Attachment 2 to this appendix, which uses a queue example, provides a simple example of a complete package with both the package specification and the package body. The parcel abstraction package is imported for use in the queue example.

These examples of packages demonstrate all of the software engineering principles:

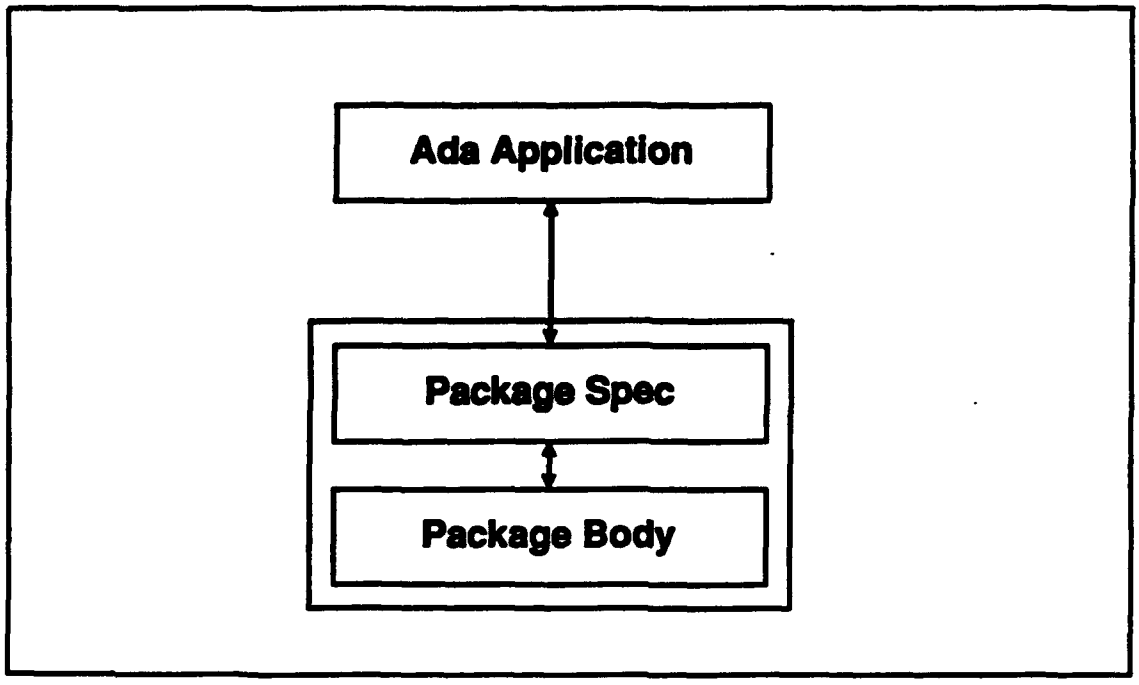

Figure L-1. Ada Package

- *Abstraction*. The package provides an excellent mechanism to create abstract data types that map to the real world. The objects and operations identified for the parcel post abstraction in the example in Attachment 1 support the requirements of the application clearly. This reduces logic errors in implementing the package body, and more important, in using the abstraction in the main program. Ada packages support data abstraction, which allows the creation of objects that correspond to real-world entities. The result is maintainable systems and the generation of code that can be reused.

- *Information Hiding*. The unnecessary detail of how the package is used is hidden from the application. This hiding prevents the application from accessing internal data structures. The package specification serves as a clean interface to the package body and hides all data structures within the body. This hiding prevents a programmer from directly accessing the data structures, which can cause two serious problems:

  - *Violation of Data Integrity*. The first serious problem is that the integrity of the data could be violated. For example, a programmer could decide that an object to be placed on the First In First Out (FIFO) queue has high priority. Attachment 2 provides a queue example. Instead of using the desired

ENQUEUE procedure, the programmer adjusts the front and back pointers in the queue, placing the new item at the front of the queue. When done incorrectly, this could destroy the integrity of the database. As a "hack," this violation would typically not be documented and not be adequately tested. If a legitimate requirement exists to place objects at the top of the queue, a special procedure should be designed as part of the package to provide this capability.

- *Undocumented, Untested Interface.* The second serious problem associated with directly accessing the data structures is that this direct access provides an undocumented coupling to the data structures that would not be updated should the package body be updated. An update to the data structures to improve performance, add new functionality, or correct an error could result in having code somewhere else in the application that no longer can work as intended. At best, this code may have no effect on the data structure. At worst, this code may totally destroy the information maintained in the data structure and invalidate it for other use. This scenario is exactly what happened in 1992 to the code that controlled the switching circuits for the telephone lines to New York City and most of New England. The misplacement of queue pointers caused the telephone system to crash for many hours. This would not have happened had the application been coded in Ada with effective use of packages. Fortunately, in Ada, the only interface to the package body is through the package specification. Consequently, the Ada package is a highly important feature for high-quality, reliable code that results in reduced costs during initial software development and later during life-cycle maintenance.

- *Completeness.* The package body can be easily tested to verify that it completely satisfies the requirements identified in the package specification. Because the only purpose of the package body is to implement the interface defined in the package specification, the package body can be easily evaluated to ensure it completely supports the interface. This minimizes the otherwise frequent surprises found during integration where requirements are not satisfied.

- *Confirmability.* The package body can be easily tested to confirm that it correctly implements the package specification. Because the only interface to the code in the package body is through the package specification, the testing problem is simplified and results in *correct* code that can be easily integrated into other compilable program units.

- *Modularity.* The package structure provides an excellent mechanism for implementing interfaces and supporting the migration to Open Systems Environments (OSEs)/Open Systems Architecture (OSA). Ada is recognized by

the National Institute of Standards and Technology (NIST) as having a strong strategic value in migrating towards OSE (U.S. Department of Commerce, 1991).

L.2 STRONG TYPING

Perhaps the second most important capability in the Ada language is the feature of ***strong typing*** coupled with the associated ***Ada exception***. Together, they provide a capability to build high-quality software by automatically identifying many programmer errors during software development at compile and execution times. For applications with reliability, fault-tolerance, and safety-critical requirements, this capability provides a mechanism to return to some known, safe state when a system error occurs. This section discusses strong typing; the next section addresses Ada exceptions.

L.2.1 Types as Building Blocks

An Ada type characterizes a set of values and a set of operations applicable to those values. Ada provides a variety of types that can be used as building blocks to create real-world abstractions. For example, Command and Control (C2) applications usually process *tracks* that represent an aircraft, ship, or submarine. Important information is maintained on each track, including identification, geographical latitude and longitude, altitude, and time of last position report. The following type definitions may be used to create a simple abstract track type:

```
type identification is (friend, foe, unknown);
type latitude is digits 12 range -90.0 .. +90.0;          —in degrees
type longitude is digits 12 range -180.0 .. +180.0;       —in degrees
type altitude is range -1000  .. +50000;                  —in feet
```

Associated with each type is a set of type-specific operations (e.g., addition and multiplication for integers and reals). These types can be used as building blocks for compound user-defined types such as arrays and records. The record is used to define the following simple logical track type:

```
type track_type is
    record
        ID:     identification;    —track ID
        lat:    latitude;          —track latitude
        long:   longitude;         —track longitude
        alt:    altitude;          —track altitude
```

```
        time:    calendar.time;   —time track position
                                  —last updated
                                  —time imported from package calendar
                                    where it is defined
    end record;
```

L.2.2 Creation of Objects From Types

A type is only a template from which objects can be created with a known set of values and a known set of operations. Objects can now be created from the above type definitions:

```
X1,X2: latitude;
Y1,Y2: longitude;
A,B:   track_type;
```

These objects can now be assigned values such as:

```
X1:= 57.0;          —X1 becomes 57.0 degrees North
X2:= X1 - 60.0;     —X2 becomes 3.0 degrees South
Y1:= -145.0;        —Y1 becomes 145.0 degrees West

A:= (friend, X1, Y1, 32_000, calendar.clock);
    —ID becomes "friend"
    —lat becomes the value of X1
    —long becomes the value of Y1
    —alt becomes 32,000 feet
    —time becomes current time (result of function clock in package
      calendar)
```

An alternative method of expressing this last assignment statement clearly associates the component objects of A:

```
A:=   (ID   => friend,          —ID becomes "friend"
       lat  => X1,                —lat becomes the value of X1
       long => Y1,                —long becomes the value of Y1
       alt  => 32_000,            —alt becomes 32,000 feet
       time => calendar.clock);   —time becomes current time
```

This method improves the understandability of the code for both programmers and nonprogrammers.

L.2.3 Handling of Objects in Homogeneous and Heterogeneous Environments

Once objects have been defined, the use of these objects as a single entity facilitates use within the application, for example:

B := A; -- The objects in record B (of track_type) are set to those of record A. This is equivalent to:

```
B.ID   :=  A.ID;
B.lat  :=  A.lat;
B.long :=  A.long;
B.alt  :=  A.alt;
B.time :=  A.time;
```

This convenient notation provides the most efficient means for handling the object within homogeneous computing environments for assignments, bus transfers, Input/Output (I/O), and other operations. A *pack/unpack* facility provides support for interfacing the object to heterogeneous computing environments. In this way, the object can be handled efficiently with one's own computer. When the object is ready to be communicated to a different computer system, it can be "packed" into the agreed-upon interface or message format.

L.2.4 Elimination of Illegal Expressions and Assignment Statements

Strong typing eliminates errors by preventing illegal expressions and illegal assignments of different types at compile time. For example, what is the result of adding five apples to six oranges? In normal mathematical situations, this is undefined. Hence, the following is undefined in reality and, in Ada, would be declared illegal at compile time:

```
X1+Y1    —illegal expression
         —adding a latitude to a longitude is undefined

Y2:=X1;  —illegal assignment statement
         —assigning a latitude to a longitude is also undefined
```

The prevention of illegal expressions and illegal assignments at compile time reduces many common logic errors found in most other languages. Although adding latitude to longitude is normally undefined and undesired, the user may choose to define such an operation in Ada.

L.2.5 Elimination of Constraint Errors at Compile Time

In addition, strong typing eliminates errors by providing constraint checking to ensure that all range values associated with the type definition are satisfied. For example, objects of type latitude are assigned to range from -90 degrees South to +90 degrees

North. In normal mathematical situations, a value outside of this range would have no meaning. Hence, the following statements would be illegal:

```
X1 := 127.0;    —illegal as 127 degrees exceeds the range constraint of 90
                 degrees North
```

L.2.6 Elimination of Constraint Errors at Run Time

Constraint errors may be identified at compile time and also at run time (during program execution). The following statement is legal for values of X1 less than and equal to 75 degrees; it is illegal when X1 is greater than 75 degrees:

```
X2 := X1 + 15.0;  —possibly illegal—only known at run time
```

Constraint checking during run time is important during software development and testing because it allows errors to be easily detected and code corrected or handled, as appropriate. Constraint checking is also important during execution in the mission environment. Should errors be detected, an exception can be raised that allows the software to take the appropriate action.

L.3 EXCEPTIONS

Ada exceptions were included in the language to support reliability, fault tolerance, and safety critical requirements. During the execution of a program, all sorts of errors can occur that could result in grave consequences. A zero divide could cause a computer to crash during critical terrain-following maneuvers; an out-of-bound index to a database could cause the entire database to be corrupted; an out-of-bound index to an array could cause a weapon to be launched against friendly forces; and an exceeded capacity limit could cause a weapon to miss the target. Errors can result from hardware faults, network faults, capacity limits, or software logic. Some errors are easy to predict; others are next to impossible. Regardless of the cause of the error, the exception feature in Ada provides an excellent mechanism to programmatically recover and return to some known, safe state and continue processing. This is important for many applications where the mission would be at risk if the computer had to be shut down and rebooted.

Without exceptions, programmers would have to test for each possible error condition and would occasionally miss a possible error. In Ada, a set of predefined exceptions exists that can automatically identify typical processing errors. It is also possible for the user to define additional error conditions that can be detected. When an error condition is detected, an ***exception*** is raised. Should an exception handler be defined for the exception, an appropriate action could be taken to return the program to a known safe state. If an exception handler is not defined, the program will crash just as a FORTRAN or C program does.

Examples of predefined exceptions are shown with the parcel abstraction example in Attachment 1, the queue example in Attachment 2, and the queue generic example in Attachment 3.

L.4 GENERICS

Generics are the building blocks of reusable software systems. Reuse is not only important for economies across applications but also can be very important within a single application.

The example of the queue, presented in Attachment 2, can be a necessary artifact to many portions of a single application. The queue, as presented in the attachment, is not very applicable for general use. It is only useful for objects of type PARCEL_TYPE going to a queue containing a maximum of 100 objects. In the past, such a queue could only be reused by hard coding the desired type and size. Making the necessary changes by hard coding such code is extremely error prone when code is complex or nontrivial. Ada provides an elegant solution. This queue can be made useful to other requirements in the same application by converting it into a generic. A generic provides a template from which new Ada code can be built. Conversion to a generic requires minor changes to the package specification and package body, some generic parameters, and a generic instantiation. A generic instantiation is a formal Ada construct that creates a logical instance of the generic code by filling in the template with the generic parameters.

The queue example in Attachment 2 has been converted to a generic queue example in Attachment 3. This example establishes generic parameters for (1) the type of item to be managed by the queue and (2) the size of the queue. The example shows the generic instantiation necessary to create an instance equivalent to the queue of Attachment 2. It also exemplifies the way this generic queue can be used to create a queue for any type of any size (up to system limits).

The power of this generic queue is considerable. Queues can now be built for any desired type for any desired size and used over and over again even within the same application. This generic can be instantiated to process parcels for shipping, radar messages, E-mail messages, financial data, stock quotes, or any data type desired and for any quantity up to hardware limits.

Once the generic is built and thoroughly tested, the cost of reusing the code is significantly reduced. Most errors will be detected and corrected when the code is first developed. This means that the cost and risk to a subsequent user will be less than that of developing the code from scratch.

Furthermore, reuse of code results in higher-quality applications. As the code is reused and corrected for each instantiation, fewer defects will be found by subsequent users. Corrections made by subsequent users can be reapplied to an earlier application during the next upgrade.

L.5 Ada LIBRARY (SEPARATE COMPILATION)

Ada provides a library mechanism that supports integration and programming-in-the-large requirements. Library units, such as package specifications, package bodies, and main programs, are managed separately. Consequently, each library unit can be compiled separately when the package specifications are known. This is important for developing large systems. Such an application can be divided among many developers by defining appropriate interfaces using the Ada package. Dummy code or stubs can be used to simulate the interface for testing and prototyping purposes. Later during integration, the completed, developed code can be very easily integrated because all portions of the code were developed by using the same interfaces.

The package body can be separately compiled from the package specification. This also supports integration by reducing the time necessary to rebuild a complete system. In the past, should an error be found in the system, the entire system had to be recompiled and linked. This frequently took days. Most execution errors are typically found in the detailed implementation in the package body. In Ada, when such errors are corrected, only the package body needs to be recompiled and the system relinked. Because this should be a very small part of the system, a complete, recompiled system can be generated rather quickly.

In addition, procedures and functions can be compiled either as part of a library unit or separately. This provides a considerable amount of flexibility when developing the design of an application and supporting early prototypes.

L.6 Ada TASKING

Ada tasking provides a capability to support logical parallel processing within an Ada application. The Ada tasking model provides an excellent and portable capability to maintain separate threads of control, synchronize asynchronous activities when necessary, and communicate among these separate threads of control. Tasking is a rather advanced language feature not found in other languages. When tasking is used, a special Run-Time Environment (RTE) is evoked to schedule tasks, process interrupts, and provide other services. It can be highly valuable to a wide variety of applications including real-time applications, simulation, prototyping, and networking. To use tasking effectively, one must understand the Ada tasking model and have a design methodology that supports the model. It is recommended that compilers be carefully evaluated because some Ada compiler implementations provide far superior support for tasking than others.

The Ada tasking model may be inappropriate for some applications because the overhead to support logical parallel processing may not justify the benefits obtained. Many organizations find that interrupt-driven sequential processing satisfies all requirements, and tasking is unnecessary. When the application is hosted on an operating system or executive, it may be practical not to use tasking in favor of the run-time provided by the environment. Common sense should prevail as to whether the Ada tasking model is appropriate for a given application.

The Ada tasking model for Ada 9X will be enhanced to directly support parallel processing with parallel processors and highly distributed environments. Section 7.1 provides additional information on Ada 9X.

L.7 FEATURES THAT FACILITATE SOFTWARE ENGINEERING

The above-described features of packages, strong typing, exceptions, generics, separate compilation, and tasking are important facilitators to software engineering. In addition, there are many other features in the Ada language, too numerous to detail here, including subtypes, access types, attributes, representation clauses, I/O, visibility, and program libraries. Although the Ada language is a cornerstone of software engineering, supporting quality, cost, and schedule benefits, Ada is only a facilitator. One must be educated and trained to use Ada with software engineering. Section 8 of Volume I provides guidance for obtaining the necessary education and training within an organization. Without the knowledge and skills to use these Ada-provided software engineering features, a programmer is likely to write programs in the same style of other languages, resulting in code with the same inefficiencies of other languages.

Attachment 1
Example—Package Specification: Parcel Abstraction Example

A simple example of an Ada package can be demonstrated with an automated post office example. Suppose the post office built a system to automatically process parcels for shipping. The design for such a system may be object oriented with abstractions for the parcel, the customer, money collection, and parcel routing. A package could be created for each of these abstractions with a main program to control the overall processing requirements. Each package would have a package specification and a package body.

The following shows an example of an Ada package specification. It is an abstraction of a parcel for shipping in the post office system. This abstraction provides a physical description of the length, width, height, and weight of the parcel. It includes shipping data on origination post office, destination post office, and method of shipment. It includes operations on the parcel such as get physical data, get shipping data, and compute shipping cost.

The main program may use the operations provided through the following code:

```
GET_PARCEL_PHYSICAL_DATA (physical_data);
GET_PARCEL_SHIPPING_DATA (shipping_data);
COMPUTE_PARCEL_SHIPPING_COST (physical_data, shipping_data,
shipping_cost);
```

The Ada package specification is an interface between the main program and the code that does the real work. The body of the PARCEL_POST package would contain the necessary code to implement the operations of GET_PARCEL_PHYSICAL_DATA, GET_PARCEL_SHIPPING_DATA, and COMPUTE_PARCEL_SHIPPING_COST. The exceptions INVALID_ZIP_CODE, PARCEL_EXCEEDS_WEIGHT_LIMITS, and PARCEL_EXCEEDS_SIZE_LIMITS may be raised when error conditions are detected. The exception INVALID_ZIP_CODE exception would be raised when an invalid destination (or origin) zip code is detected for the parcel. The exception PARCEL_EXCEEDS_WEIGHT_LIMITS would be raised when the parcel exceeds the maximum limit of 50 pounds. The PARCEL_EXCEEDS_SIZE_LIMITS would be raised when the parcel exceeds the post office size limits. In each case, application interfacing with the parcel abstraction would handle the exceptions. In the case of the excessive weight and size limits, the application may tell the customer that the parcel is rejected and return it to the customer. In the case of an invalid zip code, the application may request another zip code from the customer. The package specification for the parcel abstraction is as follows:

```
package PARCEL_ABSTRACTION is
      type INCHES is new float;
      type POUNDS is new float;
      type PARCEL_PHYSICAL_DESCRIPTION is
            record
                  LENGTH:        INCHES;          --length of parcel in inches
                  WIDTH:         INCHES;          --width of parcel in inches
                  HEIGHT:        INCHES;          --height of parcel in inches
                  WEIGHT:        POUNDS;          --weight of parcel in pounds
            end record;
      type MODE_OF_SHIPMENT is (SURFACE, AIR);    --shipping options
      type ZIP_CODE is new integer range 0 .. 99999;    --standard postal zip code
      type PARCEL_SHIPPING_DESCRIPTION is
            record
                  FROM:          ZIP_CODE;        --origination post office
                  TO:            ZIP_CODE;        --destination post office
                  SHIPMENT:      MODE_OF_SHIPMENT;
            end record;
      type DOLLAR is new float;          --cost of shipping parcel post in dollars
      type PARCEL_TYPE is
            record
                  physical_data:    PARCEL_PHYSICAL_DESCRIPTION;
                  shipping_data:    PARCEL_SHIPPING_DESCRIPTION;
                  shipping_cost:    DOLLAR := 0.0;
            end record;

      procedure GET_PARCEL_PHYSICAL_DATA (
            physical_data: out PARCEL_PHYSICAL_DESCRIPTION);

      procedure GET_PARCEL_SHIPPING_DATA (
            shipping_data: out PARCEL_SHIPPING_DESCRIPTION);

      procedure COMPUTE_PARCEL_SHIPPING_COST (
            physical_data: in    PARCEL_PHYSICAL_DESCRIPTION;
            shipping_data: in    PARCEL_SHIPPING_DESCRIPTION;
            shipping_cost: out   DOLLAR);

      exception: INVALID_ZIP_CODE;
      exception: PARCEL_EXCEEDS_WEIGHT_LIMITS;
      exception: PARCEL_EXCEEDS_SIZE_LIMITS;

end PARCEL_ABSTRACTION;
```

Attachment 2
Package Specification and Package Body: Queue Example

A simple example of a complete package can be demonstrated with the implementation of a queue. A queue is a frequently used software mechanism to buffer (or synchronize) data from one process to another. It is also known as a FIFO buffer. It is similar in concept to a queue of customers waiting to be serviced at a bank. Basically, the queue has two operations: enqueue and dequeue. When a new customer arrives, the customer enters the queue or is "enqueued." When the customer finally reaches the bank teller, the bank teller takes the customer off of the queue or "dequeues" the customer for processing.

A package specification for a queue for data of type PARCEL (defined in Attachment 1, the parcel abstraction example) capable of holding 100 objects is described in the following example. Such a queue may be used in a distributed automated application to process parcels for routing to their destination post office. Please note that the package PARCEL_ABSTRACTION is imported for use by the QUEUE package through the "with" and "use" clause on the first line.

There are two exceptions. The UNDERFLOW exception is raised when there is an attempt to dequeue an object and the queue is empty. The response here for an exception handler could be to process something else and come back to process objects in this queue later. The OVERFLOW exception is raised when the capacity limits are exceeded, in this case 100 objects in the queue. When there is an attempt to enqueue the 101st object into the queue, there is no space for the new object. In other languages, data are usually lost. In Ada, the exception handler could preserve the data and cause the process dequeuing objects off the queue to have a higher priority.

```
with PARCEL_ABSTRACTION; use PARCEL_ABSTRACTION;

package QUEUE is
     procedure ENQUEUE (parcel_object: in PARCEL_TYPE);
          —enqueues parcel_object of type PARCEL_TYPE onto queue
     procedure DEQUEUE (parcel_object: out PARCEL_TYPE);
          —dequeues parcel_object of type PARCEL_TYPE from queue
     UNDERFLOW: exception;
          —exception raised when queue is empty
     OVERFLOW: exception;
          —exception raised when capacity limits are reached
end QUEUE;
```

The package body to support such a package specification may look like:

```
package body QUEUE is

    queue: array (0 .. 99) of PARCEL_TYPE;
        --note name queue is overloaded
    front: natural := 0;          --front of the queue
    back: natural :=0;            --back of the queue

    procedure DEQUEUE (parcel_object: out PARCEL_TYPE) is
    begin
        if front = back then
            raise UNDERFLOW;
        else
            parcel_object:= queue(front);
            front:= (front+1) mod 100;
        end if;
    end DEQUEUE;

    procedure ENQUEUE (parcel_object: in PARCEL_TYPE) is
    begin
        if (back+1) mod 100 = front then
            raise OVERFLOW;
        else
            back:= (back+1) mod 100;
            queue(back):= parcel_object;
        end if;
    end ENQUEUE;

end QUEUE;
```

Procedures DEQUEUE and ENQUEUE would be used by the application using the package with parameters for an object A of type PARCEL_TYPE as:

```
ENQUEUE (A);          --enqueues object A
DEQUEUE (A);          --dequeues object A
```

Attachment 3
Generic Package: Generic Queue Example

This example demonstrates the use of Ada generics. For ease of comparison, this example provides a generic capability to the queue provided in Attachment 2. To convert the queue presented in Attachment 2 to a generic, the size of the queue and a place-holder for the type are established as generic parameters immediately before the package specification:

```
generic
     SIZE : positive;                  —any positive to be instantiated
     type ANY_TYPE is private;         —ANY_TYPE to be instantiated

package QUEUE is
     procedure DEQUEUE (any_object: out ANY_TYPE);
     procedure ENQUEUE (any_object: in ANY_TYPE);
     UNDERFLOW: exception;
     OVERFLOW: exception;
end QUEUE
```

Both the SIZE and ANY_TYPE would be provided later as parameters. The package specification has been modified to reflect the new type ANY_TYPE:

The package body is modified to reflect both the new type ANY_TYPE and the queue SIZE:

```
package body QUEUE is

     type table is array (positive range < >) of ANY_TYPE;
     queue: table(0..(SIZE-1));
     front: natural := 0;
     back:  natural :=0;

     procedure DEQUEUE (any_object: out ANY_TYPE) is
     begin
          if front = back then
               raise UNDERFLOW;
          else
               any_object:= queue(front);
               front:= (front+1) mod SIZE;
          end if;
     end DEQUEUE;
```

```
        procedure ENQUEUE (any_object: in ANY_TYPE) is
        begin
            if (back+1) mod SIZE = front then
                raise OVERFLOW;
            else
                back:= (back+1) mod SIZE;
                queue(back):= any_object;
            end if;
        end ENQUEUE;

    end QUEUE;
```

To instantiate the queue for a size of 100 objects of type PARCEL_TYPE, the following generic instantiation is made:

```
package PARCEL_QUEUE is new QUEUE (100, PARCEL_TYPE);
```

To instantiate the queue for a size of 1,000 objects of type TRACK_TYPE, the following generic instantiation is made:

```
package TRACK_QUEUE is new QUEUE (1000, TRACK_TYPE);
```

Procedures DEQUEUE and ENQUEUE are used as above, the calling application not even knowing that these procedures are from an instantiated package. With A being an object of type PARCEL_TYPE and B being an object of type TRACK_TYPE, then the following operations can be made:

```
ENQUEUE (A);        --enqueues object A into the PARCEL_QUEUE
ENQUEUE (B);        --enqueues object B into the TRACK_QUEUE
DEQUEUE (A);        --dequeues object A from the PARCEL_QUEUE
DEQUEUE (B);        --dequeues object B from the TRACK_QUEUE
```

Appendix M

Supplementary Reading

This appendix lists publications, including Software Engineering Institute (SEI) reports on data rights, that are useful to Program Managers. Reports that have Defense Technical Information Center (DTIC) numbers are available from DTIC and the National Technical Information Service (NTIS) at the following addresses:

DTIC Defense Technical Information Center
Attn.: FDRA Cameron Station
Alexandria, VA 22304-6145

NTIS National Technical Information Service
U.S. Department of Commerce
Springfield, VA 22161

SEI reports that have a DTIC number (i.e., ADA followed by six digits) may be obtained directly from:

Software Engineering Institute
Attn.: Publications Requests
Carnegie-Mellon University
Pittsburgh, PA 15213-3890

SEI affiliates and Governmental organizations may order documents directly from SEI by submitting a written request, accompanied by a mailing label with the requestor's address, to the above address.

Data Rights Reports

Martin, A. and K. Deasy. *Seeking the Balance Between Government and Industry Interests in Software Acquisition. Volume I: A Basis for Reconciling DOD and Industry Needs for Rights in Software* (CMU/SEI-87-TR-13, ADA185742). Pittsburgh, PA: Carnegie-Mellon University, 1987.

Martin, A. and K. Deasy. *The Effect of Software Support Needs on DOD Software Acquisition Policy: Part 1: A Framework for Analyzing Legal Issues* (CMU/SEI-87-TR-2, ADA178971). Pittsburgh, PA: Carnegie-Mellon University, 1987.

Samuelson, P. *Understanding the Implications of Selling Rights in Software to the Defense Department: A Journey Through the Regulatory Maze* (SEI-86-TM-3, ADA175166). Pittsburgh, PA: Carnegie-Mellon University, 1986.

Samuelson, P. *Comments on the Proposed Defense and Federal Acquisition Regulations* (SEI-86-TM-2, ADA175165). Pittsburgh, PA: Carnegie-Mellon University, 1986.

Samuelson, P. *Adequate Planning for Acquiring Sufficient Documentation About and Rights in Software to Permit Organic or Competitive Maintenance* (SEI-86-TM-1, ADA175167). Pittsburgh, PA: Carnegie-Mellon University, 1986.

Samuelson, P. and K. Deasy. *Intellectual Property Protection for Software* (SEI-CM-14-2.1). Pittsburgh, PA: Carnegie-Mellon University, 1989.

Samuelson, P., et al. *Proposal for a New "Rights in Software" Clause for Software Acquisitions by the Department of Defense* (CMU/SEI-86-TR-2, ADA182093). Pittsburgh, PA: Carnegie-Mellon University, 1986.

Appendix N

Comparison of Ada to Assembly: F-15 Structural Filter Example

Coding in High Order Languages (HOLs), such as Ada, has important benefits when compared to coding in Assembly. These benefits were demonstrated for the computation of the Structural Filter as part of the F-15 integrated flight control system, which was flown in September 1984. The formula for the S-Plane Representation of the Structural Filter was:

$$\frac{0.4807S^2 + 83.5533S + 3894}{S^2 + 125S + 3894}$$

This formula converts to the difference equation representation of:

```
STFL = 0.56503 * PRESTRU - 0.33991 * PREM1STRU + 0.089533 *
       PREM2STRU + 0.87711 * STFM1 - 0.19182 * STFLM2
```

The Ada representation of the difference equation is nearly equivalent:

```
STFL := 0.56503 * PRESTRU - 0.33991 * PREM1STRU + 0.089533 *
        PREM2STRU + 0.87711 * STFM1 - 0.19182 * STFLM2;
```

The only necessary changes are the ":=" for the assignment and the semicolon to terminate the statement. Contrast this with the Assembly version that was in the previous version of the F-15:

```
LDL   RR10,PCAS24;      % RR8 Contains PRESTRU
CALL  FMUL;             % 0.56503 * PRESTRU
LDL   RR6, RR8;         % Store Result for Later Use
LDL   RR8, PREM1STRU;
LDL   RR10, PCAS25;
CALL  FMUI;             % 0.3391 * PREM1STRU
LDL   RR10,RR8;         % Prepare for Subtraction
LDL   RR8,RR6;
CALL  FSUB;             % RR6-[0.33991 * PREM1STRU]
LDL   RR6,RR8;          % Store Result for Later Use
LDL   RR8,PREM2STRU;
LDL   RR10,PCAS26;
```

```
CALL  FMUL;          % 0.089533 * PREM2STRU
LDL   RR10,RR6;
CALL  FADD;          % RR6 + [0.089533 * PREM2STRU]
LDL   RR6, RR8;      % Store Result for Later Use
LDL   RR8,STFLM1;
LDL   RR10,PCAS27;
CALL  FMUI;          % 0.87711 * STFLM1
LDL   RR10,RR6;
CALL  FADD;          % RR6 + [0.87711 * STFLM1]
LDL   RR6,RR8;       % Store Result for Later Use
LDL   RR8,STFLM2;
LDL   R10,PCAS28;
CALL  FMUI;          % 0.19182 * STFLM2
LDL   RR10,RR8;      % Prepare For Subtraction
LDL   RR8,RR6;
CALL  FSUB;          % RR6 - [0.19182 * STFLM2]
LDL   STFI,RR8
CALL  FMUL;          % 0.089533 * PREM2STRU
```

This Assembly example uses a floating-point algorithm; had a fixed-point one been used, it would have been twice as long.

LIST OF ACRONYMS AND ABBREVIATIONS

AAS	Advanced Automation System
ABET	Ada-Based Environment for Test
ACEC	Ada Compiler Evaluation Capability
ACES	Ada Compiler Evaluation System
ACM	Association for Computing Machinery
ACSE	Association Control Service Element
ACUE	Aircraft Control Unit Emulator
ACVC	Ada Compiler Validation Capability
AdaIC	Ada Information Clearinghouse
AdaIC BB	Ada Information Clearinghouse Bulletin Board
AdaJUG	Ada Joint (Services) Users Group
Ada PSE	Ada Programming Support Environment
ADP	Automatic Data Processing
AES	Ada Evaluation System
AFATDS	Advanced Field Artillery Tactical Data System
AFB	Air Force Base
AFCEA	Armed Forces Communications and Electronics Association
AFDSRS	Air Force Defense Software Repository System
AFSC	Air Force Systems Command
AFSPACECOM	Air Force Space Command
AI	Artificial Intelligence
AIE	Ada Integrated Environment
AIS	Automated Information System
AIU	Acoustic Interface Unit
AJPO	Ada Joint Program Office
ALS	Ada Language System
ALS/N	Ada Language System/Navy
AMMWS	Advanced Millimeter Wave Seeker
AMPS	Advanced Message Processing System
ANSI	American National Standards Institute
AP	Acquisition Plan
AP	Arithmetic Processor
APB	Acquisition Program Baseline
API	Application Programming Interface
APID	Application Programming Instructional Department
APP	Application Portability Profile
APT	Advanced Programming Technique
ARB	Acquisition Review Board
ARLB	Ada Reuse Library Browser

ARPA	Advanced Research Projects Agency
ARTX	Ada Run-Time Executive
ASEET	Ada Software Engineering Education and Training
ASI	Application Software Interface
ASIS	Ada Semantic Interface Specification
ASP	Acquisition Strategy Plan
ASR	Ada Software Repository
ASSET	Asset Source for Software Engineering Technology
AST	Advanced Systems Technology
ASW	Anti-Submarine Warfare
ASWSOW	Anti-Submarine Warfare Standoff Weapon
AT&T	American Telephone & Telegraph
ATCCS	Army Tactical Command and Control System
ATD	Aircrew Training Device
ATE	Automated Test Equipment
ATF	Advanced Tactical Fighter
ATIP	Ada Technology Insertion Program
ATIS	A Tool Integration Standard
ATRIM	Aviation Training and Readiness System
AVF	Ada Validation Facility
BAFO	Best and Final Offer
BBS	Bulletin Board System
BMS	Broadcast Message Server
BP	Backplane
C2	Command and Control
C2I	Command, Control, and Intelligence
C3I	Command, Control, Communications, and Intelligence
C4I	Command, Control, Communications, Computers, and Intelligence
CAB	Common Ada Baseline
CACM	
CAD	Computer-Aided Design
CAI	Computer-Aided Instruction
CAIS	Common Ada PSE Interface Set
CALS	Computer-aided Acquisition and Logistics Support
CAM	Computer-Aided Manufacture
CAMP	Common Ada Missile Packages
CARDS	Central Archive for Reusable Defense Software Program
CASE	Computer-Aided Software Engineering
CAS REPS	Casualty Reporting System

CAUWG	Commercial Ada Users Working Group
CAXI	Common Ada XWindow Interface
CC&I	Command, Control, and Intelligence
CCITT	International Consultative Committee for Telegraph and Telephone
CCP	Code Counting Program
CCS	Combat Control System
CDA	Central Design Activity
CDB	Central Data Base
CDIF	CASE Data Interchange Format
CDPA	Central Design Programming Activity
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CECOM	Communications Electronics Command
CERT	Computer Emergency Response Team
CERT/CC	Computer Emergency Response Team Coordination Center
CFE	Contractor-Furnished Equipment
CGI	Computer Graphics Interface
CGM	Computer Graphics Metafile
CI	Configuration Item
CIF	Central Issue Facility
CIM	Corporate Information Management
CLNP	Connectionless Network Protocol
CLOC	Compiled/Assembled Lines of Code
CMM	Capability Maturity Model
CMP	CoMPleteness
CMS-2	Compiler Monitor System-2
CMU	Carnegie-Mellon University
CMU/SEI	Carnegie-Mellon University/Software Engineering Institute
CNO	Chief of Naval Operations
COBOL	Common Business Oriented Language
COE	Common Operating Environment
COEA	Cost and Operational Effectiveness Analysis
COMNAVCOMTELCOM	Commander, Naval Computer and Telecommunications Command
COMSPAWARSYSCOM	Commander, Space and Naval Warfare Systems Command
CONOPS	Concept of Operations
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off-The-Shelf
CPDL	Computer Program Development Laboratory
CPP	Command Program Processor
CPS	Competitive Prototyping Strategy

CPU	Central Processing Unit
CRADA	Cooperative Research and Development Agreement
CREASE	Catalog of Resources for Education in Ada and Software Engineering
CRISD	Computer Resource Integrated Software Document
CRLCMP	Computer Resources Life-Cycle Management Plan
CRM	Computer Resources Management
CRSS	C3I Reusable Software System
CRWG	Computer Resources Working Group
CSC	Computer Sciences Corporation
CSCI	Computer Software Configuration Item
CSRO	Center for Software Reuse Operations
CSS	Centralized Structure Store
CSS	Computer Sciences School
CSU	Computer Software Unit
CWG	Coordinator Working Group
D&V	Demonstration & Validation
DAB	Defense Acquisition Board
DACS	Data and Analysis Center for Software
DAR	Defense Acquisition Regulations
DARPA	Defense Advanced Research Projects Agency
DAT	Digital Audio Tape
DBMS	Database Management System
DC	Device Coordinate
DCDS	Distributed Computing Design System
DCE	Distributed Computing Environment
DCP	Decision Coordinating Paper
DDI	Directorate of Defense Information
DDN	Defense Data Network
DDR&E	Director of Defense Research and Engineering
DDRS	DOD Data Repository System
DEI	Data Elements in the Source
DEM	Digitized Electronic Module
DEMVAL	Demonstration and Validation
DFCS	Digital Flight Control System
DFU	De Facto Usage
DID	Data Item Description
DISA	Defense Information Systems Agency
DMRD	Defense Management Review Decision
DOD	Department of Defense
DODD	Department of Defense Directive

DODI	Department of Defense Initiative
DON	Department of the Navy
DPI	Data Processing Installation
DP/DGU	Distributed Processor/Display Generator Unit
DRPM	Direct Reporting Program Manager
DS	Directory Service
DSRS	Defense Software Repository System
DTC 2	Desk Top Computer 2
DTN	Data Transfer Network
DTIC	Defense Technical Information Center
DUS	Design Unit Specification
DWS	Defensive Weapon System
ECCM	Electronic Counter-Countermeasures
ECLD	Embedded Comment Lines in Data
ECLS	Embedded Comment Lines in Source
ECM	Electronic Countermeasures
ECMA	European Computer Manufacturing Association
ECS	Electronic Customer Services
EDI	Electronic Data Interchange
EDL	Event-Driven Language
EDSI	Equivalent Delivered Source Instructions
EMPM	Electronic Manuscript Preparation and Markup
EMR	Extended Memory Reach
ENB	Engineering Notebook
EPROM	Erasable Programmable Read Only Memory
EP	Enhanced Processor
ERA	Entity Relationship Attribute
ESD	Electronic Systems Division
ESM	Electronic Support Measure
4GL	Fourth Generation Language
FAA	Federal Aviation Administration
FAR	Federal Acquisition Regulations
FAU	Fin Actuator Unit
FCDSSA	Fleet Combat Direction System Support Activity
FD	Functional Description
FE	Functional Element
FFP	Firm Fixed Price
FFRDC	Federally Funded Research and Development Center
FIFO	First In First Out

FIPS	Federal Information Processing Standards
FIT	Flight Instrument Trainer
FMSO	Fleet Material Support Office
FP	Function Point
FPI	Functional Process Improvement
FRAWG	Front Range Ada Working Group
FSD	Full-Scale Development
FTAM	File Transfer, Access, and Management
FTP	File Transfer Program
ftp	File Transfer Protocol
43RSS	AN/UYK-43(V) Run-Time Support System
GAO	General Accounting Office
GB	Gigabyte
GEU	Guidance Electronics Unit
GFE	Government-Furnished Equipment
GFS	Government-Furnished Software
GIS	Geographic Information System
GKS	Graphical Kernel System
GM	Global Memory
GNCP	Guidance, Navigation, and Control Program
GNMP	Government Network Management Profile
GOSIP	Government Open Systems Interconnection Profile
GOTS	Government-Off-the-Shelf
GPEF	Generic Package of Elementary Functions
GPPF	Generic Package of Primitive Functions
GPO	Government Printing Office
GRACE™	Generic Reusable Ada Components for Engineering
GSIS	Graphics System Interface Standard
GTRIMS	Ground Controller Training System
GUI	Graphical User Interface
HOL	High Order Language
HP	Hewlett-Packard
HP VUE	Hewlett-Packard Visual User Environment
HPBP	High-Performance Backplane
HPP	High-Performance Processor
IBM	International Business Machines
I-CASE	Integrated Computer-Aided Software Engineering
ICC	Irvine Compiler Corporation
ICE	Independent Cost Estimate

IDEF	Integrated System Definition Language
IEC	International Electro-Technical Committee
IEEE	Institute of Electrical and Electronics Engineers
IGES	Initial Graphics Exchange Specification
IGRV	Improved Guard Rail Five
ILS	Integrated Logistics Support
ILSP	Integrated Logistics Support Plan
IMU	Inertial Measurement Unit
INEL	Idaho National Engineering Laboratory
INFOSEC	Information System Security
InProc	In Processing
I/O	Input/Output
IOC	Initial Operating Capability
IOP	Input/Output Processor
IPO	Information Planning and Organizing
IPR	In-Process Review
IPS	Integrated Project Summary
IPSE	Integrated Project Support Environment
IRAC	International Requirements and Design Criteria
IRDS	Information Resource Dictionary System
IRM	Information Resources Management
IRS	Interface Requirements Specification
IS	Information System
ISA	Instruction Set Architecture
ISC	Input Signal Conditioner
ISDN	Integrated Services Digital Network
ISEA	In-Service Engineering Activity
ISEE	Integrated Software Engineering Environment
ISO	International Organization for Standardization
ISSC	Information System Software Center
ITPB	Information Technology Policy Board
ITS	Integrated Test Software
IV&V	Independent Verification and Validation
JCS	Joint Chiefs of Staff
JIAWG	Joint Integrated Avionics Working Group
JIEO	Joint Interoperability and Engineering Organization
JLC-JPCG-CRM	Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resources Management
JTC	Joint Technical Committee

K	1,000
KAPSE	Kernel Ada Programming Support Environment
LAN	Local Area Network
LCM	Life-Cycle Management
LCSA	Life-Cycle Support Activity
LOC	Level of Consensus
LRFP	Logistics Requirements Funding Plan
MAPSE	Minimal Ada Programming Support Environment
MAT	MATurity
MB	Megabyte
MCCDC	Marine Corps Combat Development Command
MCCR	Mission-Critical Computer Resources
MCCRES	Marine Corps Combat Readiness Evaluation System
MCO	Marine Corps Order
MENS	Mission Element Need Statement
MEPS	Message Edit Processing System
MHS	Message Handling Service
MIL-HDBK	Military Handbook
MIL-STD	Military Standard
MIMMS	Marine Corps Integrated Maintenance Management System
MIPS	Millions of Instructions per Second
MIS	Management Information System
mm	Millimeter
MMI	Man-Machine Interface
MMS	Minimum Mode Software
MOA	Memorandum of Agreement
MOTS	Military Off-The-Shelf
MSE	Master's in Software Engineering
MT	Mission Trainer
NA	Network Adaptor
NAC	Naval Avionics Center
NADC	Naval Air Development Center
NAPI	North American Portable Common Tool Environment Initiative
NAPUG	North American PCTE User's Group
NARDAC	Navy Regional Data Automation Center
NASA	National Aeronautics and Space Administration
NASEE	NAVAIR Software Engineering Environment
NATO	North Atlantic Treaty Organization

NAUG	Navy Ada Users Group
NAVAIR	Naval Air Systems Command
NAVCOMTELCOM	Naval Computer and Telecommunications Command
NAVDAC	Navy Data Automation Command
NAVSEA	Naval Sea Systems Command
NAVSUP	Naval Supply Systems Command
NAVSWC	Naval Surface Warfare Center
NAWC-AD-WAR	Naval Air Warfare Center, Aircraft Division, Warminster
NCA	Naval Center for Cost Analyses
NCCOSC	Naval Command, Control, and Ocean Surveillance Center
NCS	Network Computing Service
NCTAMS	Naval Computer and Telecommunications Area Master Station
NCTAMS LANT	NCTAMS Atlantic
NCTAMS EASTPAC	NCTAMS Eastern Pacific
NCTC	Naval Computer and Telecommunications Command
NCTS	Naval Computer and Telecommunications Station
NDC	Normalized Device Coordinate
NDI	Nondevelopmental Item
NGCR	Next Generation Computer Resources
NISBS	NATO Interoperable Submarine Broadcast System
NIST	National Institute of Standards and Technology
NISMC	Naval Information System Management Center
NISO	National Information Standards Organization
NIUF	North American ISDN Users' Forum
NM	Network Management
NOSC	Naval Ocean Systems Center
NRaD	Naval Research and Development
NSWC	Naval Surface Weapons Center
NTCSS	Naval Tactical Combat Support System
NTIS	National Technical Information Service
NTSC	Navy Training and Simulation Center
NUSC	Naval Undersea Command
NWRC	Navy Wide Reuse Center
NWSUS	Navy WWMCCS Site-Unique Software
OAS	Offensive Avionics System
OASD	Office of the Assistant Secretary of Defense
OCD	Operational Concept Document
OFPS	Operational Flight Program Size
OMG	Object Management Group
OMU	Operational Mock-up

OOD	Object-Oriented Design
OOP	Object-Oriented Programming
OORA	Object-Oriented Requirements Analysis
OPE	Open Systems Environment
OPNAVINST	Naval Operations Instruction
OPR	Office of Primary Responsibility
ORG	Organization Chain of Command
OS	Operating System
OSA	Open Systems Architecture
OSD	Office of the Secretary of Defense
OSE	Open Systems Environment
OSF	Open Software Foundation
OSI	Open Systems Interconnection
OSISL	Open Systems Interface Standards List
OSS	Operations Support System
OSSWG	Operating Systems Standards Working Group
PAV	Product AVailability
PC	Personal Computer
PCIS	Portable Common Interface Set
PCTE	Portable Common Tool Environment
PDL	*Program Design Language*
PDR	Preliminary Design Review
PDS	Post-Deployment Support
PDSS	Post-Deployment Software Support
PDU	Pulse Driver Unit
PEO	Program Executive Office
PHIGS	Programmer's Hierarchical Interactive Graphics System
PII	Protocol Independent Interface
PIMB	PCTE Interface Management Board
PIWG	Performance Issues Working Group
PMC	Project Management Charter
POC	Point of Contact
POM	Program Objective Memorandum
POSIX	Portable Operating System Interface for Computer Systems
PPBS	Planning, Programming, and Budgeting System
PRISM	Portable Reusable Integrated Software Modules
PRL	PRoblems/Limitations
PRR	Product Readiness Review
PSE	Project (or Programming) Support Environment
PSERM	Project Support Environment Reference Model
PSESWG	Project Support Environment Standard Working Group

PSA	Program Structure Analysis
PSL	Program Structure Language
QA	Quality Assurance
R&D	Research and Development
RACS	Registration and Access Control System
RADC	Requirements and Design Criteria
RAM	Random Access Memory
RAPID	Reusable Ada Products for Information Systems Development
RCL	RAPID Center Library
RDA	Remote Database Access
RDBMS	Relational Database Management System
RDT&E	Research, Development, Test, and Evaluation
RES	Resources
REVIC	Revised Intermediate COCOMO
RFP	Request for Proposals
RLF	Reuse Library Framework
RLT	Reuse Library Toolset
RMA	Rate Monotonic Analysis
RMC	*Reconfigurable* Mission Computer
ROI	Return on Investment
ROM	Read Only Memory
RPC	Remote Process Communication
RPC	Remote Procedure Call
RSC	Reusable (Ada) Software Component
RTAda	Run-Time Ada
RTE	Run-Time Environment
SAE	Software Architectures Engineering
SAFENET	Survivable Adaptable Fiber-optic Embedded Network
SAI	Software Action Item
SAIL	System Avionics Integration Laboratory
SAME	SQL Ada Module Extension
SAMeDL	SQL Ada Module Description Language
SASET	Software Architecture Sizing and Estimating Tool
SASSY	Supported Activities Supply System
SCAI	Space Command & Control Architecture Infrastructure
SCCS	Submarine Combat Control System
SCE	Software Capability Evaluation
SCH	Scheduler

SCL	Stand-alone Comment Lines
SCMP	System Configuration Management Plan
SCP	System Concept Paper
SCRB	Software Change Review Board
SCS	Submarine Combat System
SDC-W	Software Development Center, Washington
SDD	System Design Definition
SDE	Software Development Environment
SDF	Software Development Folder
SDIO	Strategic Defense Initiative Organization
SDL	Software Development Laboratory
SDP	Software Development Plan
SDP	System Division Paper
SDR	System Design Review
SDSR	Software Development Status Report
SDTS	Spatial Data Transfer Standard
SECNAVINST	Secretary of the Navy Instruction
SECNAVNOTE	Secretary of the Navy Note
SECR	Standard Embedded Computer Resource
SEE	Software Engineering Environment
SEI	Software Engineering Institute
SEM	Standard Electronic Module
SEMP	System Engineering Management Plan
SEO	Software Executive Official
SEOC	Software Executive Official Council
SEPG	Software Engineering Process Group
SES	Senior Executive Service
SGS/AC	Shipboard Gridlock System with Auto-Correlation
SGML	Standard Generalized Markup Language
SIGAda	Special Interest Group on Ada
SIGSOFT	Special Interest Group on Software Engineering
SIL	System Integration Laboratory
SIP	System Integration Plan
SISTO	Software and Intelligent Systems Technology Office
SLCMP	Software Life-Cycle Management Plan
SLOC	Source Lines of Code
SLOC/SM	Source Lines of Code per Staff Month
SLOCWC	Source Lines of Code Without Comments
SMB	Submarine Message Buffer
SMM	Software Management Metrics
SMP	Software Master Plan
SOW	Statement of Work

SPA	Software Process Assessment
SPAWAR	Space and Naval Warfare Systems Command
SPC	Software Productivity Consortium
SPD	Software Process Definition
SPDL	Standard Page Description Language
SPI	Software Process Improvement
SPO	System Programming Office
SPR	Software Problem Report
SQAP	Software Quality Assurance Plan
SQL	Structured Query Language
SRC	Software Requirements Change
SRP	Software Reuse Program
SRR	Software Requirements Review
SRS	Software Requirements Specification
SSA	Software Support Activity
SSC	System Support Center
SSS	System/Segment Specification
STANFINS	Standard Financial System
STANFINS-R	Standard Financial System Redesign
STARFIARS	Standard Army Financial Accounting and Reporting System
STARS	Software Technology for Adaptable, Reliable Systems
STB	STaBility
STC	Software Technology Conference
STEP	Standard for the Exchange of Product Model Data
STI	Software Technology Initiative
STSC	Software Technology Support Center
SUP	Support Planning
SWAP	Software Action Plan
SWAP-WG	Software Action Plan Working Group
SWG	Special Working Group
SWTP	Software Technology Plan
SYSCOM	Systems Command
TAC	Tactical Advanced Computer
TACAMO	Take Charge and Move Out
TACFIRE	Tactical Fire Direction
TAFIM	Technical Architecture For Information Management
TADSTAND	Tactical Digital Standard
TAMPS	Tactical Aircraft Mission Planning System
TC	Target Capacity
TC	Technical Committee
TCL	Total Comment Lines

TCP/IP	Transmission Control Protocol/Internet Protocol
TD	Technical Directive
TDA	Technical Directive Authority
TDT	Theater Display Terminal
T&E	Testing and Evaluation
TeleAda EXEC	Telesoft Run-Time Environment
TEMP	Test and Evaluation Master Plan
TEP	Test and Evaluation Plan
TFA	Transparent File Access
TLCSC/LLCSC	Top Level/Lower Level Computer Software Component
TLOC	Total Lines of Code
TOES	Telephone Order-Entry System
TOPS	Training and Operations Section
TQM	Total Quality Management
TSGCEE	Tri-Service Group on Communications and Electronics Equipment
UIMS	User Interface Management System
ULLS	Unit Level Logistics System
USMC	U.S. Marine Corps
USTAG	United States Technical Advisory Group
USW	Undersea Warfare
UUT	Unit Under Test
VADS	Verdix Ada Development System
VDI	Virtual Device Interface
VHSIC	Very High-Speed Integrated Circuit
VRC	Virtual Reference Coordinate
VSR	Validation Summary Report
VT	Virtual Terminal
VUE	Visual User Environment
WAdaS	Washington Ada Symposium
WAM	WWMCCS ADP Modernization
WBS	Work Breakdown Structure
WC	World Coordinate
WFNIA	Wells Fargo Nikko Investment Advisors
WIS	WWMCCS Information System
WST	Weapon System Trainer
WPAFB	Wright Patterson Air Force Base
WWMCCS	World Wide Military Command and Control System